**SocialWave design**

**Juan Daniel Torres**

**Juan Manuel Diaz**

**Estructura discretas y computación (G1)**

**Jeison David Mejía Trujillo**

**Facultad de ingeniería, Ingeniería de sistemas**

**Cali, 19 de mayo de 2023**

**SocialWave design**

**Context:**

SocialWave is a social media application that aims to connect people and foster interactions within a virtual community. It provides users with a platform to share content, communicate, and build relationships with others. However, the app currently faces some challenges in terms of its functionality and user experience. The main problems in the SocialWave social network revolve around the lack of certain crucial functionalities. Firstly, there is a need for the implementation of a feature that enables users to search for mutual friends. This functionality would empower users to discover and explore their shared connections with other platform users, enhancing their social experience and facilitating new connections.

Additionally, SocialWave lacks a social influence analysis capability. This analysis involves utilizing graph centrality metrics, such as degree, closeness, or betweenness, to identify influential users within the network. By analyzing the social influence of users, the platform can devise effective marketing strategies and identify key individuals who can drive engagement and interaction within the community. Furthermore, the absence of a friendship connection recommendation system is another significant problem. Such a system would leverage algorithms like the minimum spanning tree to suggest and facilitate the establishment of new friendships between users with common interests or connections. This feature would not only improve the overall user experience but also foster a sense of community and enhance the social networking aspect of SocialWave. To address these problems, the SocialWave app needs to prioritize the development and implementation of functionalities that enable users to search for mutual friends, conduct social influence analysis, and establish a friendship connection recommendation system for providing a more satisfying and meaningful experience for its users.

**Step 1:**

**Definition of the problem**:

What is the problem?

The problem is the lack of certain functionalities in the SocialWave social network, such as the ability to search for mutual friends, conduct social influence analysis and develop a friendship recommendation system.

Who has the problem?

The users of the app have the problem, that's because they are currently unable to perform tasks such as searching for mutual friends, analyzing social influence and receiving friendship recommendations based on common interests or connections.

Why is it important to solve?

It is important to solve this problem because implementing these functionalities will improve the overall user experience of the SocialWave social network. Allowing users to search for mutual friends will help them discover and explore their shared connections, enhancing their social interactions on the platform. Conducting social influence analysis will help identify influential users, enabling effective marketing strategies. Developing a friendship connection recommendation system will facilitate the establishment of new friendships based on common interests or connections, further fostering user engagement and interaction within the social network.

Identification of needs and symptoms:

* The system should search for mutual friends between two users.
* The system needs to implement the most efficient algorithms for its operation.
* The system should allow analyzing the most influential users in the app.
* The system needs a simple interface for users.
* The system should show a specific user recommended friends according to the algorithm of the social network.

**Step 2.**

**Information Gathering**

**Graphs:**

A graph is a data structure consisting of nodes (vertices) and edges that represent connections between the nodes. It is used to model and analyze relationships or interactions between various elements. Graphs provide a powerful way to represent and work with complex networks, allowing for tasks such as finding shortest paths, analyzing connectivity, identifying cycles, and determining influential nodes. They are widely employed to solve a diverse range of problems, from representing social networks and data dependencies to optimizing transportation routes and internet routing.

**Depth First Search Algorithm:**

The DFS algorithm is a graph traversal algorithm that explores a graph by visiting nodes in a depthward motion. It starts at a chosen node and explores as far as possible along each branch before backtracking. The algorithm maintains a stack or recursion to keep track of the nodes to be visited. When visiting a node, it marks it as visited and recursively explores its unvisited neighbors. This process continues until all reachable nodes have been visited.

**Dijkstra 's Algorithm.:**

This algorithm makes a tree of the shortest path from the starting node, the source, to all other nodes (points) in the graph. Dijkstra's algorithm makes use of weights of the edges for finding the path that minimizes the total distance (weight) among the source node and all other nodes. This algorithm is also known as the single-source shortest path algorithm. It is important to note that Dijkstra’s algorithm is only applicable when all weights are positive because, during the execution, the weights of the edges are added to find the shortest path.

**Breadth First Search Algorithm:**

The BFS algorithm is a graph traversal algorithm that explores a graph by systematically visiting nodes in a breadthward fashion. It starts at a chosen node and explores all its neighboring nodes before moving to the next level of neighbors. The algorithm maintains a queue data structure to keep track of the nodes to be visited. When visiting a node, it marks it as visited and enqueues its unvisited neighbors. This process continues until all reachable nodes have been visited.

**Floyd-Warshall Algorithm:**

This algorithm is a dynamic programming algorithm used to find the shortest paths between all pairs of nodes in a weighted graph. It works by gradually building up a matrix of shortest path distances between every pair of nodes. Initially, the matrix is initialized with the direct edge weights between adjacent nodes. Then, for each pair of nodes, the algorithm considers all intermediate nodes and checks if there is a shorter path through any of these intermediate nodes. If a shorter path is found, the matrix entry for that pair of nodes is updated with the new shortest path distance. By repeatedly considering all possible intermediate nodes, the algorithm progressively computes the shortest path distances between all pairs of nodes, resulting in the final matrix of shortest path distances.

**Minimum Spanning Tree:**

This algorithm is used to find the subset of edges in a graph that connects all nodes with the minimum total edge weight. The resulting subset forms a tree with no cycles and spans all nodes of the original graph. The MST algorithm is commonly used in graph theory and has various applications, including the analysis of social networks.

**Prim 's Algorithm:**

Prim's algorithm is a well-known algorithm used to find the minimum spanning tree (MST) of a weighted graph. It starts by selecting an arbitrary node as the starting point and gradually grows the MST by greedily choosing the edge with the smallest weight that connects a node in the MST to a node outside of it. This process continues until all nodes are included in the MST. By iteratively expanding the MST and selecting the minimum-weight edges, Prim's algorithm efficiently constructs a tree that spans all nodes while minimizing the total weight. In the context of social networks, the Prim's algorithm can be applied to identify the most influential connections among users, as the included edges represent the critical relationships that contribute to the overall connectivity and influence within the network.

**Pagerank algorithm:**

The PageRank algorithm is a widely used algorithm for measuring the importance or influence of nodes in a graph, particularly in web pages. It assigns a numerical score to each node based on the concept of "voting." In the context of a graph, a node's score is influenced by the scores of its neighboring nodes. Nodes with higher scores contribute more to the scores of their neighbors, and the process continues iteratively until the scores converge. The algorithm assumes that a node is important if it is linked to by other important nodes. By iteratively calculating and updating the scores, PageRank effectively identifies influential nodes in the graph, reflecting their potential impact and significance within the network.

**Kruskal 's Algorithm:**

Kruskal's algorithm is a popular algorithm used to find the minimum spanning tree (MST) of a weighted graph. It starts by sorting the edges of the graph in non-decreasing order based on their weights. Then, it iterates through the sorted edges and adds them to the MST if they do not create a cycle. This cycle detection is typically performed using a disjoint-set data structure. By repeatedly adding edges to the MST until all nodes are connected or all edges have been considered, Kruskal's algorithm efficiently finds the subset of edges that forms the MST, minimizing the total weight. In the context of social networks, the MST algorithm can be applied to identify the most influential connections among users, as the included edges represent the crucial relationships that contribute to the overall connectivity and influence within the network.

**Uniform Cost Search Algorithm:**

The Uniform Cost Search algorithm is a graph traversal algorithm that finds the path with the minimum cost between a start node and a goal node in a weighted graph. It explores the graph by continually expanding the node with the lowest cumulative cost from the start node. The algorithm maintains a priority queue or a priority heap to keep track of the nodes to be visited. It starts at the start node with a cost of zero and explores its neighboring nodes, calculating their cumulative costs. The algorithm selects the node with the lowest cost and continues expanding from there until it reaches the goal node or exhausts all possible paths. The Uniform Cost Search algorithm ensures that the shortest path with the minimum cumulative cost is found.

**Algorithm A\* (A-Star):**

The A\* algorithm is a heuristic search algorithm used to find the shortest path between a start node and a goal node in a weighted graph. It combines the advantages of both Dijkstra's algorithm (which guarantees the shortest path) and greedy best-first search (which uses heuristics to prioritize exploration). The algorithm maintains a priority queue or a priority heap to keep track of the nodes to be visited. It evaluates nodes based on two values: the cost to reach the current node from the start node (known as g-value) and an estimated cost to reach the goal node from the current node (known as h-value). The A\* algorithm selects the node with the lowest sum of g-value and h-value and continues expanding from there until it reaches the goal node or exhausts all possible paths. The h-value is typically estimated using heuristics such as Euclidean distance or Manhattan distance. By using an informed heuristic approach, the A\* algorithm efficiently explores the graph, finding the shortest path to the goal node.

**Step 3.**

**Search for Creative Solutions**

For finding the best solution that we´re going to implement, brainstorming was the technique we implemented in order to get the most ideas out of us.

**Brainstorming**

**R1. Mutual Friends Search**

**Depth First Search Algorithm (DFS):**

To implement the DFS algorithm for finding mutual friends within an influence graph, we would follow these steps: Select a starting person as the source node. Create an empty stack to keep track of visited nodes. Push the starting person onto the stack and mark it as visited. While the stack is not empty, pop a person from the top of the stack. Iterate through their friends, and if a friend has not been visited, push it onto the stack and mark it as visited. Check if the friend is already a mutual friend. If not, add it to the list of mutual friends. Continue this process until the stack is empty. Finally, return the list of mutual friends. By exploring the graph in a depth-first manner using a stack, this implementation effectively finds the mutual friends of people within the influence graph.

**Breadth First Search Algorithm (BFS):**

To implement the BFS algorithm for finding mutual friends within an influence graph, we would start by selecting a person as the source node. then create a queue and a set to track visited nodes. Enqueue the source node and mark it as visited. While the queue is not empty, dequeue a person and iterate through their friends. If a friend has not been visited, enqueue it, mark it as visited, and check if it is already a mutual friend. If not, add it to the list of mutual friends. Continue this process until the queue is empty. Finally, return the list of mutual friends. By exploring the graph in a breadth-first manner, this implementation should find the mutual friends of people.

**Uniform Cost Search Algorithm:**

As this algorithm is not typically used for finding mutual friends within an influence graph. It is primarily used for finding the shortest path between nodes in a weighted graph. However, for us to adapt the algorithm for this purpose, we would assign a uniform cost of 1 to each edge in the influence graph and modify the goal condition to include checking for mutual friends. We would start with a source person and explore their friends, enqueuing them with a cumulative cost of 1. As you explore each friend, you can check if they are already a mutual friend and add them to the list if not. By continuing the search until the goal condition is met or all reachable nodes are explored, then we would be able to find the mutual friends of a person using a modified version of the Uniform Cost Search algorithm.

**Algorithm A\* (A-Star):**

This algorithm is not directly applicable for finding mutual friends within an influence graph since it is primarily used for pathfinding in weighted graphs. However, adapting it to solve this problem by assigning appropriate heuristics and modifying the goal condition. We would need to define a heuristic that estimates the distance or similarity between two people based on their shared connections or interests. Starting with a source person, then prioritizing exploration based on the combined cost of reaching a person and the estimated similarity to the goal of mutual friendship. As the graph is being traversed, we would check if a person is a mutual friend and add them to the list if not. By iteratively expanding the most promising nodes until the goal condition is met or all reachable nodes are explored, we could potentially find mutual friends using a customized version of the A\* algorithm, anyways would not be a viable method.

**R2. Social influence analysis**

**Dijkstra 's Algorithm:**

To implement the Dijkstra algorithm for social influence analysis within the given problem context, we would first construct a weighted graph representing the social network. Each node in the graph would represent a user, and the edges would represent their connections. The weights of the edges could represent the strength or influence of the relationship between users.

Then we would initialize the algorithm choosing a starting node, once all nodes have been visited, we should have the shortest path distances from the starting node to all other nodes in the graph. These distances can be used to compute centrality metrics such as degree, closeness, or betweenness. Degree centrality represents the number of connections a user has, closeness centrality measures how easily a user can reach others in the network, and betweenness centrality quantifies the extent to which a user lies on the shortest paths between other users.

Then for the friendship connection recommendation system using the minimum spanning tree algorithm, we could utilize the resulting graph from the Dijkstra algorithm. The MST algorithm constructs a tree that connects all nodes with minimum total edge weight. In the context of friendship connections, we would consider the influence or relationship strength as the edge weight. By applying the MST algorithm on the graph, we should be able to identify the most important and influential connections among users, which can be used as recommendations for new friendships or effective marketing strategies.

**Floyd-Warshall Algorithm:**

To implement this algorithm for social influence analysis and develop a friendship connection recommendation system using the minimum spanning tree algorithm, we would start by constructing a weighted graph representing the social network, where users are nodes and their relationships are edges with corresponding weights indicating influence or relationship strength. Then, apply the Floyd-Warshall algorithm to find the shortest path distances between all pairs of users in the graph. These distances can be used to compute centrality metrics such as degree, closeness, or betweenness, enabling the identification of influential users within the network. Additionally, utilize the resulting graph to apply the minimum spanning tree algorithm, which will reveal the most important and influential connections among users, facilitating the generation of friendship recommendations and the development of effective marketing strategies based on influential relationships.

**Pagerank algorithm:**

To implement this algorithm inside the social influence requirement we would firstly initialize the social influence graph, then as the purpose of this algorithm is each node would receive a weight according to the neighbors it already has, it wouldn't be parameterized according to the distance it exists between nodes but for the importance of the neighbors ir currently has, additionally it would only be needed to run only one algorithm instead of two or more as the previous possible solutions.

**R3. Recommendation of friendship connections**

**MST algorithm:**

This algorithm is used to find the subset of edges in a graph that connects all nodes with the minimum total edge weight. The resulting subset forms a tree with no cycles and spans all nodes of the original graph. The MST algorithm is commonly used in graph theory and has various applications, including the analysis of social networks.

With this definition, we might be able to recommend friends to the users based on the influence that this node has inside the graph, despite the common links or other factors it could help them to find new important people.

**Prim 's Algorithm:**

Firstly we have to initialize the graph and its entire structure, so as the problem mentioned before computing the centrality metrics of the graph, such as degree, closeness, or betweenness, to identify the influential users within the network, then identify potential friendship connections by considering users who have common connections. This can be done by analyzing the graph structure, looking for shared neighbors, with this information we would rank the potential friendship connections based on the centrality metrics and other relevant factors. Consider the influence involved users, as well as their compatibility in terms of common friends.

Then, we would utilize the minimum spanning tree algorithm to refine the friendship connection recommendations. Construct the MST of the social influence graph, considering the weights of the edges as a measure of relationship strength or influence. The resulting MST will highlight the most significant connections among users and can be used as a basis for friendship recommendations.

Finally with this information we could provide the friendship connection recommendations to users through a user interface or notification system.

**Kruskal 's Algorithm:**

Firstly we have to initialize the graph and its entire structure, so as the problem mentioned before computing the centrality metrics of the graph, such as degree, closeness, or betweenness, then we would apply the Kruskal algorithm to find the minimum spanning tree (MST) of the graph. The MST will represent the most important and influential connections among users while minimizing the total weight. Furthermore, we would have to analyze the MST to identify potential friendship connections. Considering the nodes that are not directly connected by an edge in the MST but have common neighbors or share similar interests. These users are likely to apply for a friendship recommendation.

Finally we would present the friendship connection recommendations to users. Provide the recommendations through a user interface. Display the recommended users and the reasons behind the suggestion, such as shared connection.

**Step 4.**

**Transition from ideas to preliminary designs.**

Why did we discard the other alternatives?

**R1. Mutual Friends Search**

**Uniform Cost Search Algorithm:**

We discarded this algorithm because its main purpose is to find the minimum cost between nodes. In this requirement the need we have to solve is to search mutual friends, so the factor of weight between nodes is irrelevant. though we could use it in another requirement.

**Algorithm A\* (A-Star):**

The A\* algorithm is a heuristic search algorithm used to find the shortest path between a start node and a goal node in a weighted graph. so in order to use it to solve this requirement it's almost useless, we have to find the related nodes of the current friends the user already has.

**R2. Social influence analysis**

**Dijkstra 's Algorithm:**

Although Dijkstra's algorithm can find the shortest path between two nodes in a weighted graph, it does not focus on the overall influence of the nodes .

**Floyd-Warshall Algorithm:**

This algorithm is primarily used for finding all-pairs shortest paths in a graph, rather than quantifying social influence.

**R3. Recommendation of friendship connections**

**MST algorithm:**

If our purpose is to achieve an appreciated recommendation we might not use this algorithm, because even sometimes it could actually suggest related people, the main function it will throw is going to be for importance instead of common friends.

**Find the most efficient way to search for mutual friends.**

**Depth First Search Algorithm (DFS):**

* Performs a deep scan, allowing to find mutual friends efficiently by comparing the neighbor lists of nodes A and B (Users).
* DFS uses a stack to store nodes and branches needed during exploration, which prevents excessive memory consumption, especially on large data sets.

**Breadth First Search Algorithm (BFS):**

* BFS performs a layered search, which makes it easy to identify mutual friends by comparing the neighbor lists of nodes A and B (Users).
* BFS makes it possible to find mutual friends near and far between A and B (Users), providing a more comprehensive view of the connections and network of mutual friends at different levels of distance.

**R2. Social influence analysis.**

**Dijkstra 's Algorithm:**

* Dijkstra can be adapted to calculate proximity centrality in a graph. By determining the shortest distance between a given node and all other nodes, the algorithm identifies nodes with high proximity centrality, indicating that those users have many connections in the social network.
* By using the centrality metrics provided by Dijkstra. Those nodes with a high proximity centrality can be considered influential, since they have a greater capacity to disseminate information, influence other users or design marketing strategies on the platform.

**Floyd-Warshall Algorithm:**

* The Floyd-Warshall algorithm considers all possible paths between nodes, which makes it possible to identify users who act as bridges between different communities in the social network. This is valuable for understanding the structure of the network and designing influence strategies.

**R3. Recommendation of friendship connections.**

**Prim 's Algorithm:**

Prim's algorithm starts with an arbitrary node and greedily adds the edges with the minimum weights that connect the current MST to new nodes. In the context of friend recommendation, Prim's algorithm can be applied to identify influential users and their connections, forming a network of friendships that maximizes overall influence and connectivity. By considering the weights of the edges as a measure of influence or relationship strength, the resulting MST can guide the recommendation process, highlighting the most crucial friendships that would enhance users' social influence.

**Kruskal 's Algorithm:**

Kruskal's algorithm sorts the edges of the graph by weight and progressively adds them to the MST. In the context of friend recommendation, Kruskal's algorithm can identify potential friendships by considering connections that contribute to the overall influence and connectivity within the graph. By selecting edges with lower weights, which indicate stronger relationships or higher influence, Kruskal's algorithm helps identify influential friendships that can be recommended to users.

**Step 5.**

**Evaluation and Selection of the Best Solution.**

**R1. Mutual Friends Search**

1. Implementing Depth First Search Algorithm (Method 1):

Pros:

1. Efficiency: DFS is a relatively efficient algorithm, especially in sparse graphs, as it explores each branch of the graph depth-first, avoiding unnecessary exploration of unrelated branches.

2. Memory Efficiency: DFS has a lower memory requirement compared to other graph traversal algorithms like BFS, as it only needs to store information about the current path.

3. Simplicity: DFS is a straightforward algorithm to implement and understand, making it accessible for developers.

Cons:

1. Completeness: DFS may not guarantee finding all possible mutual friends, especially if the graph contains cycles or if there are disconnected components. It can get stuck exploring one branch and miss potential mutual friends in other parts of the graph.

2. Lack of Optimality: DFS does not guarantee finding the shortest path or the most optimal solution. It may find a mutual friend path that is suboptimal in terms of length or other metrics.

3. Lack of Breadth-First Exploration: DFS explores the graph depth-first, which means it may not consider nodes that are farther away in terms of connections. This can result in missing potential mutual friends who are further from the starting node.

1. Implementing Breadth First Search Algorithm (Method 2):

Pros:

1. Completeness: BFS guarantees finding all possible mutual friends, as it explores all nodes in the graph that are reachable from the starting node.

2. Optimality: BFS ensures finding the shortest path between nodes, which can be advantageous when searching for the shortest mutual friend path within the social influence graph.

3. Breadth-First Exploration: BFS explores the graph level by level, considering nodes at the same distance from the starting node before moving to the next level. This allows for a broader exploration of the graph and can potentially uncover mutual friends who are farther away but have direct connections.

Cons:

1. Memory Usage: BFS may require more memory compared to DFS as it needs to store information about all the nodes at a given level during the exploration. In graphs with a large number of nodes or deep levels, this can result in high memory usage.

2. Slower in Dense Graphs: In dense graphs, where the number of edges is close to the maximum possible, BFS may become slower compared to DFS due to the larger number of nodes to explore and maintain in memory.

3. Lack of Efficiency in Large Graphs: If the social influence graph is extremely large, BFS can be computationally expensive and time-consuming, especially if the goal is to find mutual friends between multiple pairs of nodes.

**Criterion A. Ease of implementation**

- [1] difficult

- [2] medium

- [3] easy

**Criterion B. Efficiency**

- [1] low

- [2] average

- [3] high

**Criterion C. Precision**

- [1] low

- [2] average

- [3] high

**Criterion D. Resource use**

- [1] low

- [2] average

- [3] high

|  | Criterion A | Criterion B | Criterion C | Criterion D | Total |
| --- | --- | --- | --- | --- | --- |
| Method 1 | 3 | 3 | 3 | 3 | 12 |
| Method 2 | 3 | 3 | 2 | 1 | 9 |

Selection of solution:

According to the above evaluation, the best solution for mutual friends search is the Breadth First Search Algorithm to solve the requirement.

**R2. Social influence analysis.**

1. Implementing dijkstra algorithm (Method 1):

Pros:

1. Optimality: Dijkstra's algorithm guarantees finding the shortest path between nodes based on the weights of the edges. In the context of social influence analysis, this can be valuable for identifying the most influential users within the graph, considering the strength or influence of their relationships.

2. Flexibility: Dijkstra's algorithm can accommodate different weightings or metrics associated with the edges of the graph. In the context of social influence analysis, it allows for the consideration of various centrality metrics (e.g., degree, closeness, betweenness) as weights, enabling a comprehensive analysis of social influence within the graph.

3. Scalability: Dijkstra's algorithm performs well in small to medium-sized graphs, making it suitable for social influence analysis in relatively moderate-sized social networks.

Cons:

1. Computational Complexity: Dijkstra's algorithm has a time complexity of O(V^2) or O(E log V) depending on the implementation. In larger graphs with many nodes and edges, the computational cost can become significant, leading to potential performance challenges.

2. Limited Applicability to Dynamic Graphs: Dijkstra's algorithm is designed for static graphs where edge weights remain constant. In dynamic social networks where relationships and influences change over time, the algorithm may need to be adapted or combined with other approaches to provide accurate and up-to-date social influence analysis.

3. Memory Usage: Dijkstra's algorithm requires storing and updating information about distances and visited nodes, which can consume a significant amount of memory, especially in larger graphs.

1. Implementing Floyd\_Warshall algorithm (Method 2):

Pros:

1. All-Pairs Shortest Path: The Floyd-Warshall algorithm computes the shortest path between all pairs of nodes in a graph. In the context of social influence analysis, this allows for a comprehensive analysis of the influence relationships between all users in the graph.

2. Efficiency in Dense Graphs: The Floyd-Warshall algorithm has a time complexity of O(V^3), making it efficient for dense graphs where the number of edges is close to the maximum possible. It can handle larger graphs with reasonable computational resources.

3. Handling Negative Edge Weights: Unlike other shortest path algorithms like Dijkstra's algorithm, the Floyd-Warshall algorithm can handle graphs with negative edge weights. This flexibility can be useful in scenarios where negative influences or relationships are considered in the social influence analysis.

Cons:

1. Memory Usage: The Floyd-Warshall algorithm requires storing and updating a distance matrix for all pairs of nodes, which can be memory-intensive, especially for large graphs. The space complexity of the algorithm is O(V^2), which can be a limitation in memory-constrained environments.

2. Inefficiency in Sparse Graphs: In graphs with a low density of edges, where the number of edges is significantly smaller than the maximum possible, the Floyd-Warshall algorithm can be less efficient compared to other algorithms designed specifically for sparse graphs.

3. Lack of Scalability to Large Graphs: The time complexity of the Floyd-Warshall algorithm makes it less suitable for very large graphs with a large number of nodes. The algorithm's performance can deteriorate rapidly as the graph size increases, making it impractical for analyzing social influence in extremely large networks.

**Criterion A. Ease of implementation**

- [1] difficult

- [2] medium

- [3] easy

**Criterion B. Efficiency**

- [1] low

- [2] average

- [3] high

**Criterion C. Precision**

- [1] low

- [2] average

- [3] high

**Criterion D. Resource use**

- [1] low

- [2] average

- [3] high

|  | Criterion A | Criterion B | Criterion C | Criterion D | Total |
| --- | --- | --- | --- | --- | --- |
| Method 1 | 2 | 3 | 3 | 3 | 11 |
| Method 2 | 3 | 2 | 3 | 2 | 10 |

Selection of solution:

According to the above evaluation, the best solution for mutual friends search is Dijkstra's Algorithm to solve the requirement.

**R3. Recommendation of friendship connections.**

1. Implementing prim's algorithm(Method 1):

Pros:

1. Minimum Spanning Tree: Prim's algorithm guarantees the creation of a minimum spanning tree, which ensures that the recommended friendship connections are optimal in terms of the total influence or weight within the graph. This promotes efficient and effective recommendations.

2. Efficiency: Prim's algorithm has a time complexity of O(E log V) when implemented using a binary heap, making it efficient for moderate-sized graphs. It can handle larger graphs with reasonable computational resources.

3. Flexibility: Prim's algorithm allows for the incorporation of edge weights or influence metrics as a factor in the selection of friendship connections. This flexibility enables the consideration of social influence and connectivity strength in the recommendation process.

Cons:

1. Limited Exploration: Prim's algorithm focuses on constructing a minimum spanning tree rooted at a particular node. As a result, it may not explore all possible friendship connections or uncover connections that are outside the scope of the minimum spanning tree.

2. Single Starting Point: Prim's algorithm requires a starting node to initiate the construction of the minimum spanning tree. Depending on the choice of the starting node, the resulting friendship recommendations may vary, potentially overlooking other potential connections.

3. Lack of Adaptability to Dynamic Graphs: Prim's algorithm assumes a static graph, where edge weights or influences remain constant. In dynamic social networks where relationships and influences change over time, the algorithm may need to be adapted or combined with other approaches to provide up-to-date and relevant friendship recommendations.

1. Implementing Kruskal algorithm(Method 2):

Pros:

1. Minimum Spanning Tree: Kruskal's algorithm constructs a minimum spanning tree, which ensures that the recommended friendship connections have the minimum total influence or weight in the graph. This promotes efficient and effective recommendations.

2. Exploration of Multiple Paths: Kruskal's algorithm considers all available edges in the graph, allowing for the exploration of multiple paths and potential friendship connections. It does not rely on a single starting point, providing a more comprehensive view of the graph's connectivity.

3. Flexibility: Kruskal's algorithm can handle graphs with both positive and negative edge weights, making it suitable for situations where the influence relationships can have varying strengths or directions.

Cons:

1. Efficiency in Dense Graphs: Kruskal's algorithm has a time complexity of O(E log E) when implemented using a suitable data structure, such as a union-find data structure. In dense graphs with a large number of edges, the algorithm's computational cost can be significant.

2. Memory Usage: Kruskal's algorithm requires storing and processing all the edges in the graph, which can be memory-intensive, especially for large graphs. The space complexity of the algorithm is O(E), which may pose memory constraints in certain environments.

3. Limited Consideration of Social Influence: Kruskal's algorithm primarily focuses on finding a minimum spanning tree based on edge weights. While it guarantees an optimal solution in terms of total influence or weight, it may not take into account specific social influence metrics or centrality measures that could be important for friendship recommendations.

**Criterion A. Ease of implementation**

- [1] difficult

- [2] medium

- [3] easy

**Criterion B. Efficiency**

- [1] low

- [2] average

- [3] high

**Criterion C. Precision**

- [1] low

- [2] average

- [3] high

**Criterion D. Resource use**

- [1] low

- [2] average

- [3] high

|  | Criterion A | Criterion B | Criterion C | Criterion D | Total |
| --- | --- | --- | --- | --- | --- |
| Method 1 | 2 | 3 | 2 | 2 | 9 |
| Method 2 | 3 | 3 | 3 | 3 | 12 |

Selection of solution:

According to the above evaluation, the best solution for mutual friends search is the Kruskal Algorithm to solve the requirement.

**Bibliography:**

Graphs: <https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure>

Depth First Search Algorithm: <https://www.javatpoint.com/depth-first-search-algorithm>

Dijkstra 's Algorithm: <https://www.analyticssteps.com/blogs/dijkstras-algorithm-shortest-path-algorithm>

Breadth First Search Algorithm: <https://www.simplilearn.com/tutorials/data-structure-tutorial/bfs-algorithm>

Floyd-Warshall Algorithm: <https://brilliant.org/wiki/floyd-warshall-algorithm/>

Minimum Spanning Tree: <https://www.statisticshowto.com/minimum-spanning-tree/>

Prim 's Algorithm: <https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/>

Pagerank algorithm: <https://towardsdatascience.com/pagerank-algorithm-fully-explained-dc794184b4af>

Kruskal 's Algorithm: <https://www.simplilearn.com/tutorials/data-structure-tutorial/kruskal-algorithm>

Uniform Cost Search Algorithm: <https://www.educba.com/uniform-cost-search/>

Algorithm A\* (A-Star): <https://brilliant.org/wiki/a-star-search/>