Social Media Project

<u>Given:</u>

We are trying to create a social media model using unweighted undirected graphs to represent social relationships within the social media.

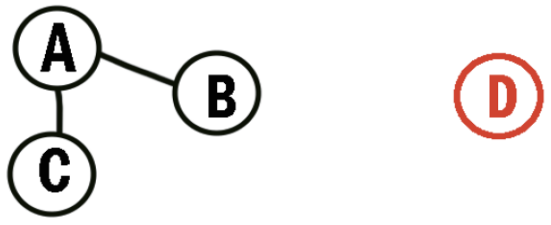
In the graph, nodes represent people while edges represent mutual “friendships” (i.e. they follow each other)

<u>Goals:</u>

Through our model we want to provide the following services:

1. Add nodes (people) to the social network

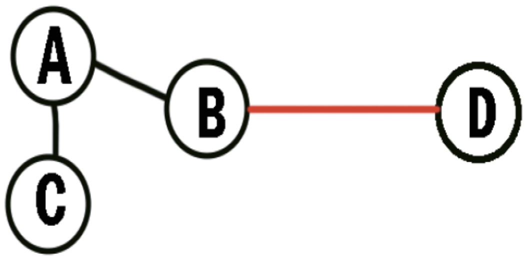
We want to be able to add people to our social network, this is similar to creating a new account with 0 friends.



<u>Example 1: D is the new node added to the graph with the 0 new connections.</u>

2. Make two nodes friends

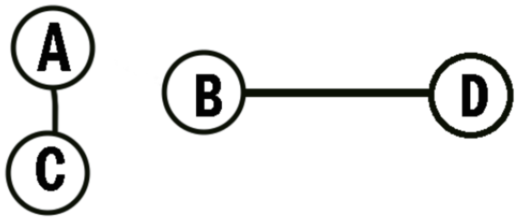
We want accounts to be able to friend each other. This is similar to being mutual friends with someone online.



Example 2: A new edge is created between *B* and *D* signifying they are mutual friends now

3. Unfriend some people from the social network

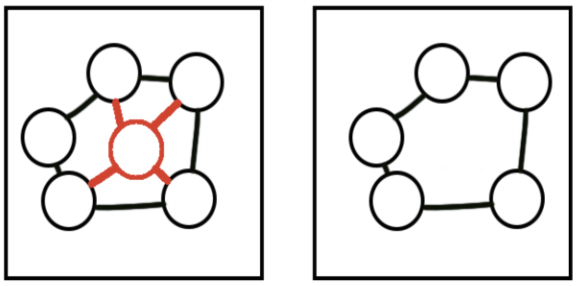
We want to allow people to unfollow each other. This is similar to unfriending someone.



Example 3: *A* and *B* are no longer friends.

4. Remove someone from the social network

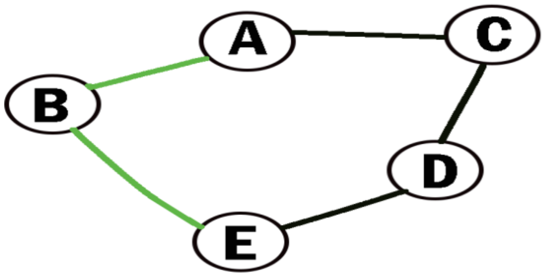
Social media platforms allow people to delete their accounts, we want to have the same functionality. Deleting someone's account entails removing the connections they have with everyone else.



Example 4: The middle node deleted their account so they got removed from the graph with all their connections.

5. Provide a node *A* the shortest path to communicate to a node *B* through friends of friends

The social media network provides a function that allows users to pass messages from one person to another even if they aren't friends. However, this message must be propagated through friends of friends. To do this efficiently we want to find the shortest path of friends from node *A* to node *B*.

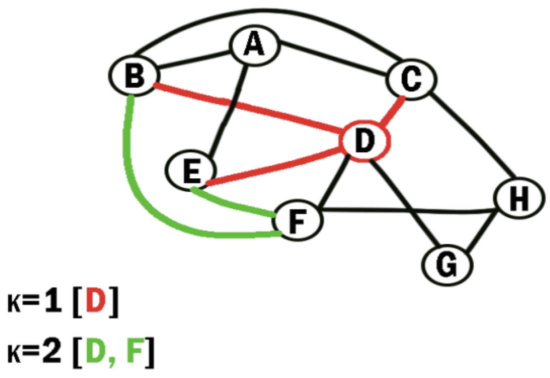


Example 5: Person A wants to send a message to person E and the shortest path is A -> B -> E

If no path exists return an empty path.

6. Recommend to a node *A* the top *k* people to follow based on the number of mutual friends they have.

Our social media should be able to recommend people to become friends based on a metric. Our metric will be the number of mutual friends the two nodes have. We want to be able to recommend more than 1 person though, so our social media should be able to return any number *k* of nodes that are the nodes with the highest number of mutual friends between nodes *A* and *B*.



Example 6: k=1 and we want to recommend k nodes to form a friendship with node *A*. Here we recommend node *D* because it has the most friends in common with A. If k=2 we would've recommended *D* and *F* since D has 3 mutual friends and F has 2.

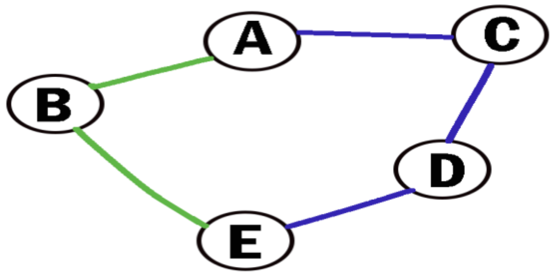
7. Confirm whether a node *A* and a node *B* are friends.

We want to check whether nodes *A* and *B* are friends.

[no example]

8. Provide a node *A* the shortest path to communicate to a node *B* while avoiding specific nodes

This is similar to goal 5 but with an added challenge. The social media network provides a function that allows users to pass messages from one person to another even if they aren't friends. However, this message must be propagated through friends of friends. To do this efficiently we want to find the shortest path of friends from node *A* to node *B*. **This time however we want to be able to avoid certain nodes that the user inputs in a blacklist, so if node A was hosting a party and didn't B to attend but B was in the shortest path then we calculate another shortest path that doesn't include B.**



Example 7: The normal shortest path from A to E is through B but if B was blacklisted then the next shortest path is A->C->D->E.

Note that the blacklist can include more than one node. If no path exists return an empty path.

Notes:

- Paths should be represented by vectors.
- The graph should be represented as an edge list.
- The recommended friends list is represented by a vector.