When a user searches for a person ("target") on an online social network, the user might want to be a friend of the target. How would you design a system to help the user?

HW6 explores graph algorithms that can help the user finds mutual friends or "intermediate" friends of the user and the target. A mutual friend is someone who is both a friend of the user and the target. For example, if areFriends(user, Mickey) and areFriends(Mickey, target), Mickey is a mutual friend. An intermediate friend is someone who is a friend of the user and is indirectly connected to the target in the social network. For example, if areFriends(user, Mickey), areFriends(Mickey, x), areFriends(x, y), ..., and areFriends(z, target), Mickey is an intermediate friend. Naturally, we also desire the intermediate friend who is closest to the target. The user can ask the mutual or intermediate friend to introduce the user to the target. Moreover, users can add friendships :-) and remove friendships :-(.

To find mutual or intermediate friends, we can use the breadth-first search algorithm. The algorithm allows us to find the shortest path from the user to the target. If the path is of length 2, a mutual friend exists. If the path length is more than 2, an intermediate friend exists. Note that multiple paths of the same length might exist and a path might not exist at all. If at least one shortest path exists, the system will report the shortest path(s) and the mutual/intermediate friend(s).

HW6 Extra Credit (via HW6Extra.java) [20 points]: To estimate how close a friendship is, we can measure the frequency of communication (e.g. texts, email, calls, ...) between two users. Frequency of communication in HW6 is the average number of days between two successive communications and is an integer (for simplicity). Using Dijkstra’s shortest path algorithm, we can find the mutual or intermediate friend on the shortest path from the user to the target. 10 points for reporting the shortest path (ties are arbitrarily broken). 10 points for reporting multiple shortest paths if ties exist.

Input: Command-line argument for HW6.java (HW6Extra.java) is:

- filename of initial friendships—the first line has the number of users, each of the following lines has two users who are friends (followed by frequency of communication for the Extra Credit problem)
- filename of actions—possible actions (each on one line) are:
  - AddFriendship user1 user2 (frequency in Extra Credit)
  - RemoveFriendship user1 user2
  - WantToBefriend user target

For simplicity, all the users are in the initial friendships and the number of users does not change during the actions. Assume users are valid in the actions.

Output: Output goes to the standard output (screen):

1. AddFriendship user1 user2 (frequency in Extra Credit) [ExistingFriendshipError]
2. RemoveFriendship user1 user2 [NoFriendshipError]
3. WantToBefriend user target [AlreadyAFriendError]
   Length of shortest path: k
   You have n mutual/intermediate friends in p paths — try asking them for introductions:
   x — path is user x y ... z target
   ...
   or
   Sorry, none of your friends can help introduce you to target.

Sample input and output files are on the course website.

Submission: Submit HW6.java that has the main method and other program files. Submissions for Individual and GroupHelp have the same guidelines as HW1.

For Extra Credit, submit HW6Extra.java that has the main method and other program files. GroupHelp and late submissions are not applicable.

Note the late penalty on the syllabus if you submit after the due date and time as specified at the top of the assignment.