CSE 2010, HW6

Due Tue Apr 11 at the start of your lab section; Submit Server: class = cse2010, assignment = hw6SxIndividual
Due Tue Apr 11 at the end of your lab section; Submit Server: class = cse2010, assignment = hw6SxGroupHelp
x is 14, 23, or c—your merged section number (c for C lang.).

When a user searches for a person ("target") on an online social network, the user might want to be a friend of the target, but the user is not currently a friend of the target. For example, a user would like Joe Biden to be his/her friend, but the user is not a friend of Joe Biden. How would you design a system to help the user?

HW6 explores graph algorithms that can help the user find 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, Jon) and areFriends(Jon, target), Jon 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, Jon), areFriends(Jon, x), areFriends(x, y), ..., and areFriends(z, target), Jon is an intermediate friend. Naturally, we also desire the intermediate friend who is closest to the target. The user can ask the mutual/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).

Friends (adjacent vertices) are visited in alphabetical order (to produce unique output for easier testing/debugging). We will be evaluating your submission on code01.fit.edu; we recommend you to ensure that your submission runs on code01.fit.edu. To preserve invisible characters, we strongly recommend you to download and save, NOT copy and paste, input data files.

Extra Credit 1: (10 points) Separate submission via HW6Extra1.java (or hw6extra1.c). Similar to regular credit, except if ties exist, multiple shortest paths are reported (in the order when adjacent vertices are visited alphabetically).

Extra Credit 2: (30 points) Separate submission via HW6Extra2.java (or hw6extra2.c). Not all friendships are equally close. Closer friendships are more desirable in finding the target. 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. Similar to regular credit, report the first shortest path found—adjacent vertices are visited alphabetically.

Extra Credit 3: (10 points) Separate submission via HW6Extra3.java (or hw6extra3.c). Similar to Extra Credit 1, except if ties exist, multiple shortest paths are reported (in the order when adjacent vertices are visited alphabetically).

Input: Command-line argument for HW6.java (hw6.c) [and Extra Credit] 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 communcation for Extra Credits 2 and 3)
- filename of actions—possible actions (each on one line) are:
  - AddFriendship *user*1 *user*2 [*frequency* in Extra Credits 2 and 3]
  - RemoveFriendship *user*1 *user*2
  - 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 *user*1 *user*2 (*frequency* in Extra Credit) [*ExistingFriendshipError*]
2. RemoveFriendship *user*1 *user*2 [*NoFriendshipError*]
3. WantToBefriend *user target* [*AlreadyAFriendError*]
   - Length of the shortest path: *k*
   - Your *mutual/intermediate* friend is *x*.
   - Path: *user x y ... z target*
   or
   - Sorry, none of your friends can help introduce you to *target*.

Extra Credit 1: Only WantToBefriend is different with possibly additional mutual/intermediate friends and paths. The following 2 lines are repeated for each additional mutual/intermediate friend:
- Your *mutual/intermediate* friend is *x*.
- Path: *user x y ... z target*

Extra Credits 2 and 3: Same as the corresponding output for Regular Credit and Extra Credit 1.

Sample intput and output files are on the course website.

Submission: Submit HW6.java (hw6.c) that has the main method and other program files. Submissions for Individual and GroupHelp have the same guidelines as HW1.

For Extra Credits, submit HW6Extra1.java, HW6Extra2.java, and/or HW6extra3.java (or hw6extra1.c, hw6extra2.c, and/or hw6extra3.c) that have 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.