Computer Science 271 – Fall 2017
Assignment #4

Due: Wednesday, 12/06/17 at 11:59 PM in D2L

In this assignment, you will implement a social network using a graph and then analyze the graph. Ideally, you should use the Graph ADT you are working on in labs 11 and 12 as your building blocks, but I don't require that. You can also incorporate other ADTs from the Java `Collections` API.

We will represent a friendship network as an undirected, weighted graph with persons as vertices, represented by their string user ids. An edge between person `A` and person `B` means that `A` considers `B` a friend, and that `B` considers `A` a friend. The weight on the edge is the intensity of their friendship (1-3), with 3 indicating a close friend, 1 a casual acquaintance, and 2 in between.

1. Implement this code so that you can create and populate a friendship network from a file in which every line is an edge in the format below:
```
                person1 person2 intensity
```
   Only edges are printed in this file. You must deduce the vertices and set up all weighted edges from this information. *Prompt for this file name*.

2. Report who the person(s) with the largest number of friends in this network is, and how many friends he/she has.

3. *Prompt for user-ids of two persons*. Then report the distance, and path between these two persons. For this problem, consider every friendship as having a weight of 1 and compute the shortest path between the friends. Here we are simply counting the number of connections/links between these two persons. If a path does not exist, simply report so.

4. *Prompt for a specific user id* and then provide a ranked list of five recommended friends (or less, if 5 recommendations are not available) for this person. Provide these recommendations in two ways:
   a. *Friends-of-friends* Recommendation: Implement the rule-of-thumb that states that the best friend recommendation is the person with whom you have the largest number of mutual friends. To implement this, consider the friends of all your friends. For each of them, compute a *mutuality* score by summating the intensities of friendship each of you have with your common friends. Rank these friends of friends by descending mutuality scores and report the top 5. For example, assume A and B have C, D and E as friends in common. Let the friendship intensity for A be C_A, D_A and E_A, and B be C_B, D_B and E_B with these friends respectively. Then the mutuality score for recommending B as a friend of A is computed as $(C_A+C_B+D_A+D_B+E_A+E_B)$, with a higher score indicating a higher recommendation.

b. *Influence* Recommendation: Consider the following hypothetical situation. Two of your friends are X and Y. X has only two friends (you and one other person). Y has a million friends. X and Y have no friends in common besides you. Since X is highly selective in terms of friendship, and is a friend of yours, you are likely to have a lot in common with X's other friend. On the other hand, Y is indiscriminate and there is little reason to believe that you should be friendly with any particular one of Y's other friends.

   We will ignore friendship intensities for this score and incorporate the above idea as follows: Suppose that A and B have three friends in common: C, D, and E. Now the influence score for recommending B as a friend of A is

   `1/numfriends(C) + 1/numfriends(D) + 1/numfriends(E)`,

   where `numfriends(F)` is the number of friends that F has. In other words, each friend F of A has a total influence score of 1 to contribute, and divides it equally among all of F's friends. Again, rank these persons by descending influence scores (with a higher score indicating a higher recommendation) and report the top 5.

Two example files, `RJ.txt` and `friends.txt`, are available on D2L in the zipped file `A4.zip`.

Your code must be cleanly designed and well documented. Please submit a zipped folder containing your NetBeans project folder to the D2L drop box `Assignment 4` by the specified deadline.