CSCI2110
Friend Recommendation

January 29, 2024

Note: This specification is provided as part of Assignment 2, 3 and 4 of CSCI2134 in Winter 2024 and explains what the code is supposed to do. Please see the assignment instructions for your task, e.g. writing unit tests for the code. Do not spend time writing code for the friend recommendation problem.

The purpose of this assignment is to get you to create and use maps and graphs, and further improve your programming skills.

As discussed in class and the first tutorial, for each problem you will be provided with a description of the problem and a JUnit test class to test your code. A similar JUnit test class will be used to evaluate your code. You can use Eclipse or other IDEs to run the JUnit test class to test your code.

Your code **must** compile. If it does not compile, you will receive a 0 on the assignment.

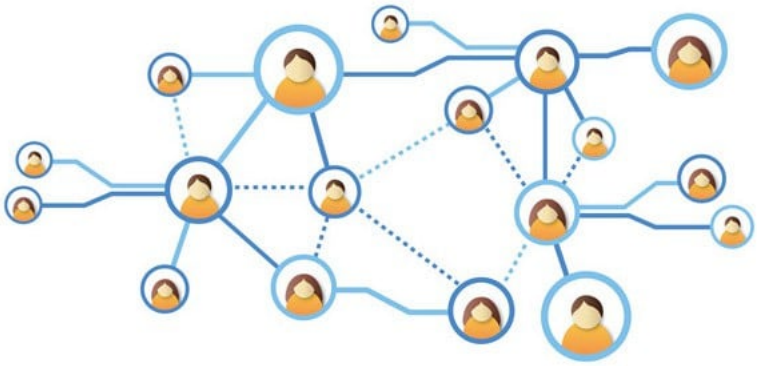

Figure 1: `www.stockphotosecrets.com/free-photos/free-vectors/social-network-grid.html`

Background: Social Networks

Social networks are networks that represent relationships between people. One common relationship is "friend". This relationship is bidirectional, e.g., if Alice is friends with Bob, then Bob is friends with Alice. Social networks are constantly changing as users join or leave the network, and friend or unfriend other users. Many social networks have a "friend recommender" feature that recommends new friends to members if they share a mutual friend. For example, if Alice is friends with Bob, and Carol befriends Bob, the system will recommend Alice and Carol to each other.

Problem: Recommend Friends

Given a series of user actions in the social network that is initially empty, recommend new possible friendships after one user friends another user. For example, if Alice, Bob, and Carol are all network members, and Alice and Bob are friends. If Bob and Carol become friends, then your program should recommend that Alice and Carol should become friends.

For example, given the user actions the resulting recommendations would be:

```
Alice joins
Carol joins
Bob joins
Bob friends Alice
Bob friends Carol
Dave joins
Dave friends Bob
end
```

Figure 2: Sample user actions.

```
Alice and Carol should be friends
Alice and Dave should be friends
Carol and Dave should be friends
```

Figure 3: Friend recommendations based on user actions in Figure 2

Your task is to create a program that reads in the user actions and generates the friend recommendations.

Write a program called `Friend.java` that reads in the user actions from the console (*System.in*) and outputs the corresponding friend recommendations. **Your `Friend` class must implement the provided `Tester` interface and also contain the `main()` method where your program starts running.** This is because your program will be tested via this interface. The interface contains a single method:

```
public ArrayList<String> compute( Scanner input );
```

This method must perform the required computation.

Input

The `compute()` method takes a *Scanner* object, which contains one or more lines. Each line, except the last one, contains a user action. There are four user actions:

`A joins` : means that new user A has joins the network.

`A friends B` : means that user A and B are now friends.
`A unfriends B` : means that user A and B are no longer friends.
`A leaves` : means that user A has left the network (deleted their account).

where A and B are names of users. The user names are single words with no white spaces within them. The last line of input is simply the word "`end`". See Figure 2 for an example.

Hint: Use the `nextLine()` method of the *Scanner* object to read one line of input at a time, and then parse the line. See the solution for Assignment 1 for how to parse input one line at a time.

Semantics

The social network is initially empty. All user actions are on the same network.

Only users that are not part of the social network can join. I.e., each user will have a unique name. Users can leave the network and then rejoin.

When a user leaves the network, the user and all his/her friendships are deleted from the network. If a user rejoins the network, it's as if he/she is joining for the first time.

All friendships are symmetrical. I.e., "`A friends B`" is the same as "`B friends A`". **No recommendation should be generated for users who are already friends.**

Output

The method `compute( Scanner input )` should return an *ArrayList* of *String*s denoting the friend recommendations in the same order as the user actions that caused them. A recommendation is of the form

A `and` B `should be friends`

where A and B are user names and **must** be in alphabetical order. For example, this recommendation is correct: "`Alice and Bob should be friends`", while this one is not: "`Bob and Alice should be friends`". If more than one recommendation is generated by a single user action (as in Figures 2 and 3), the recommendations should be ordered alphabetically.

Example

Sample Input	Sample Output

`Alice joins`
`Carol joins`
`Bob joins`
`Bob friends Alice`
`Bob friends Carol`
`Dave joins`
`Dave friends Bob`
`end`
	`Alice and Carol should be friends`
	`Alice and Dave should be friends`
	`Carol and Dave should be friends`

Hints and Suggestions

- Use a 1-pass algorithm: Simply check for friendships after processing each user action.
- There is not a lot of code to write. The sample solution is under 115 lines of code.
- Your code **must** compile. If it does not compile, you will receive a 0 on the assignment.
- Your code must be well commented and indented. Please see the Assignments section for this course on Brightspace for Code Style Guidelines.
- You may assume that all input will be correct.
- Be sure to test your programs using the provided JUnit test class.

Grading

The assignment will be graded based on three criteria:

Functionality "Does it work according to specifications?". This is determined in an automated fashion by running your program on a number of inputs and ensuring that the outputs match the expected outputs. The score is determined based on the number of tests that your program passes. So, if your program passes $\frac{t}{T}$ tests, you will receive that proportion of the marks.

Quality of Solution "Is it a good solution?" This considers whether the solution is correct, efficient, covers boundary conditions, does not have any obvious bugs, etc. This is determined by visual inspection of the code. Initially full marks are given to each solution and marks are deducted based on faults found in the solution.

Code Clarity "Is it well written?" This considers whether the solution is properly formatted, well documented, and follows coding style guidelines.

If your program does not compile, it is considered non-functional and of extremely poor quality, meaning you will receive 0 for the solution.

	Marks
Functionality	20
Quality of Solution	20
Code Clarity	10
Total	50

Table 1: Marking scheme for Friend Recommendation.

What to Hand In

Submit the source files for your program in a zip file (`.zip`). You should have at least one source file: `Friend.java`. If you are using Eclipse, we recommend that you submit the entire project bundle. Submission is to be done via Brightspace. Your code **must** compile. If it does not compile, you will receive a 0 on the assignment.