CISC-121 Winter 2013

Week 4 Lab Assignment: Social Networking

This is the assignment for the labs in Week 4 (January 28 – February 1). This assignment will not be handed in or graded.

Deadline: Please complete your solution by Monday February 4. If you fall behind, it's very hard to catch up. I will post my solution but please don't look at it until you have worked through the lab and completed your own solution. In terms of learning, looking at my solution is a very poor substitute for creating your own.

Introduction: Social networks may signal the end of civilization, but they also serve as useful sources of interesting programming problems. In this assignment we will model a very small set of people and the connections between them. The connections will represent a "Likes" relationship, which is not necessarily symmetric: it is possible for A to like B, without B liking A. Your job is to create and implement an algorithm that will discover chains of "Likes" connections between randomly chosen people in the set.

Details:

1. I have provided a Python program called **social_network.py** that creates and prints the "Likes" information for a small set of people. Download this program and study it carefully. It introduces some new functions from the **random** library, such as

   choice( ) - randomly chooses an element of a list
   sample( ) - randomly chooses several elements of a list

   The program also uses a data structure called a **dictionary.** We will talk briefly about dictionaries in class, and you should also look them up in the text. They are not difficult and in this assignment we are using a dictionary in a very simple way. Please ask me or one of the TAs for help if you have trouble understanding this part of the program.

   The program also uses the list function **remove**, which removes an item from a list. I used this because the randomly generated "Likes" information might result in a person "Liking" themselves – that's not really a problem but I thought it made sense to remove "self-likes" to simplify your task in part 2.

   At the end of the program you will see a segment of code that randomly chooses a "source" person and a "target" person, and makes sure they are not the same person. Feel free to use this for testing purposes.

2. The purpose of your program is to find a chain of "Likes" from one person to another. For example, a chain linking "Julius" to "Arwen" might be
   Julius likes Harry
   Harry likes Marion
   Marion likes Vesper
   Vesper likes Arwen

Your program should

1. either
   - prompt the user for the names of the two people for whom a chain should be found, or
   - choose two people at random
2. find a chain from the first person to the second, if such a chain exists
3. print the chain, in a format similar to the one shown above, or report that no chain exists from the first person to the second

There are very efficient algorithms to solve this problem, guaranteed to find the shortest possible chains. For this assignment, you can (and should) devise your own algorithm, and find any chain that you can. You are not required to find the shortest possible chain.

Increasing the Problem Size: The program currently has a very small set of people. Try adding enough names to double the size of the set of people. Does your program still work correctly?

Dealing with Errors: If you decide to let the user choose the two people, you need to be sure the selected names are in the list of people. One very easy way to do this is to use an EasyGUI function called **choicebox** – see the EasyGUI tutorial at http://www.ferg.org/easygui/tutorial.html for full information.

Testing: Even for a small set of people, testing your program on all possible sets of connections is infeasible. Even checking the results for all pairs of people within one set of connections is time consuming. Nonetheless, some amount of testing and verification of the output can give us confidence that the program is correct. Testing all different outcomes is very important. In this program there are two possible outcomes: either a chain of connections exists or it doesn't. Given the way the Likes connections are generated, it is very unlikely that two people will be unconnected. Yet this is possible, and your program must handle the situation properly. Consider how you can engineer the Likes connections to test the ability of your program to handle this situation.

The Rest of the Story:

This problem is not really about social networks. It is actually about finding routes through communication networks – a problem that gets solved a bazillion times every day as email and data packets of all kinds are transmitted across the Internet. In the real world

the networks are immense and the routing decisions need to be made very quickly. Fast and reliable algorithms are a must.