ECE 108 Project 3
Social Networks and the Social Graph

Bernie Roehl, ECE Department, May 2018
Based on a design by M. Tripunitara and J. Thistle

Introduction

In this lab you will be implementing a simple social network, and using it to identify *dissimilar* groups of users. This is a practical implementation of the solution to Problem 2 of Assignment 3.

Background

The social network you'll be implementing has two kinds of entities: Users and Contents. Each user and each piece of content is identified by a unique small integer, counting from 1. In other words, the first user has an id of 1, the next has an id of 2 and so on. Similarly, the first piece of content has id 1, the second id 2 and so on.

There are three kinds of relations: "is a friend of", "owns" and "likes".

Users can be friends, and friendship is *symmetric* (i.e. if someone is your friend, then you're their friend as well). Friendship is also *reflexive*, meaning that you're friends with yourself. We refer to the "is a friend of" relation as F.

Users can own pieces of content. We refer to the "owns" relation as W.

Users can like pieces of content that are owned by others. We refer to this relation as L.

The following additional rules apply:

1. If you are friends with someone, you like everything they own.
2. If you like anything that someone owns, you're *similar* to them even if you're not friends.
3. If you're not similar to someone, then you're *dissimilar* to them.

Your Task

You will be provided with an input file that specifies the number of users, the number of content items, a list of friendships, a list of content ownerships and a list of likes. You will need to read this input file and build some in-memory representation of each of the three relations (F, W and L). The format of the input file is described later in this manual.

Your goal is to find all sets of users that are dissimilar to one another. In other words, you need to go through every possible pair of subsets of users, and find those subsets that are dissimilar.

Two sets are considered dissimilar if all the individual users in the first set are dissimilar to all the users in the second set.

The Algorithm

You will be implementing the algorithm described in the solution to the assignment. The overall approach is as follows…

1. Make sure that your friendship relation is reflexive, i.e. that all users are friends with themselves. You can call this modified friendship relation F^r.
2. Make sure that your friendship relation is symmetric, i.e. that if user A is friends with user B, then user B is friends with user A. Call this F^{rs}.
3. Find any additional content that users like, in addition to the content they have explicitly stated that they like You can find this additional content using rule 1 above, in other words by finding the content owned by each person's friends. This is simply the *composition* of F^{rs} with W. Once you have found this additional content, you merge it (using a Union operation) with the explicitly-provided likes. In other words, $\mathbf{L^{fw} \leftarrow L \cup (F^{rs} \circ W)}$
4. Find users who are similar to a given user, using rule 2 above. These will be the users who own content that the given user likes. Since W tells us what content a particular person owns, the inverse of W tells use who owns a particular piece of content. You can therefore composite L^{fw} with the inverse of W to find out the owners of content that any given user likes. In other words, $\mathbf{S \leftarrow L^{fw} \circ W^{-1}}$
5. Find the dissimilarity relation D, which is just the complement of S. In other words, $\mathbf{D \leftarrow U^2 - S}$ where U is the set of all users.
6. The final step is to go through every possible combination of two subsets of U (the set of all users) and find which pairs of subsets are dissimilar to each other. The dissimilarity between a pair of sets is found by going through the cartesian product of the two sets and seeing if all the resulting pairs of users are dissimilar (i.e. are in the relation D).
7. As you find those pairs of sets, print them out in the format described below.

The algorithm given in the assignment solution refers to a set of sets called $\mathbb{D}$ that holds all the results. This is not a requirement, and you are permitted to simply output the pairs of sets as you identify them. You are also not required to explicitly create the relation D (the complement of S) if you find that you don't need to.

Note that the algorithm has a runtime that is exponential in the number of users (since the number of subsets of a set is 2^n where *n* is the number of elements in the set, and we're also using a nested loop to iterate over all pairs of subsets). There are ways of optimizing this, but they are beyond the scope of this project. You can assume that the number of users is small enough that performance shouldn't be an issue unless your implementation is exceptionally bad.

You can also assume that the empty set is never similar to any other set, including itself. You can also assume that the set of dissimilar sets is irreflexive (i.e. a set is never dissimilar to itself).

Input and Output Formats

The input consists of a series of lines. The first line is "#U" and is followed (on a separate line) by the number of users. This is followed by a "#C" and the number of content items.

Next is a "#F" followed by a series of pairs of integers (one pair per line), the first giving the id of a user and the second giving the id of one of that user's friends. Similarly, this is followed by #W and a series of pairs of integers giving a user id and the id of some content owned by that user. Finally, there's a #L followed by a series of pairs of integers giving a user id and the id of some content that the user likes.

Your output should consist of a series of lines, each containing two sets separated by a space. Each set should be a comma-separated list of user ids, and each set is surrounded by curly braces { }.

Setting Up and Submission

We provide you with a repository called SocialData that contains two files -- `input.txt`, which contains sample input in the format described above, and `expected_output.txt` which contains the results you should see. Note that your output may not necessarily be in the same order as the expected output, and that's fine.

Your submission repository should be called SocialNetwork. Your program should be called **`dis`** (for "dissimilar"). You must provide a Makefile that the TAs can use to build **`dis`** (but not run it). Your program should read from standard input and write to standard output, which may be redirected using the Linux < and > command-line operators. For example, your program can read from cin (which would normally be the keyboard) and you can redirect that input to come from a file instead.

To grade your project, the TA will check out your SocialNetwork directory from SVN and do the following:

```
make
./dis   <input.txt   >output.txt
```

They will then compare `output.txt` to `expected_output.txt` (allowing for the possible different ordering of the lines).

They will then run your code against two additional input files (which are not provided to you) and compare your output to the corresponding expected results.

Note that your program should behave reasonably in cases where the number of users and/or the number of content items is zero.

Marking

If your code produces output that matches `expected_output.txt` (aside from possibly ordering the lines differently), you get 50%. You get an additional 25% for correctly processing each of the remaining two input files.

Hints on Implementation

You should create a class that stores a relation, and provide a way of building instances of that class from the input file. You should implement a number of methods on that class, such as composition, union and so on.

One way of implementing such a class is by using adjacency matrices, as discussed in the lectures.

One of the most interesting parts of this project is the enumeration of all the possible subsets of a set. Note that there are 2^n such subsets, so one approach is to treat each integer in the range 0 to 2^n-1 as a representation of a set, with the individual bits of the integer corresponding to the presence or absence of a particular element in the set. In this representation, 0 is the empty set.

More details are provided in the slides.