Problem Set 3 Exercise #28: Friendship Relations [Hard]

Reference: Lecture 9 notes

Learning objectives: Two-dimensional array; Algorithm design

Estimated completion time: 90 minutes

Problem statement:

[CS1010 AY2011/12 Semester 1 Exam, Q6]

A local entrepreneur wishes to develop a new social network system, called *iLink* and she employs you to help develop programs to handle friendship relation service. In modelling the friendship relation, you have adopted a two-dimensional array called **friendArr**. A simplified version of **friendArr** with 6 users is given below:

	0	1	2	3	4	5
0	1	0	1	0	0	0
1	0	1	0	0	0	1
2	1	0	1	1	0	0
3	0	0	1	1	0	0
4	0	0	0	0	1	1
5	0	1	0	0	1	1

Under this representation, you set the entry (*i*, *j*) of **friendArr** to 1 if the user identified by *i* has added the user identified by *j* as a **direct friend**. Otherwise, entry (*i*, *j*) should contain 0. By default, an *iLink* user will always add himself/herself as a direct friend, and the **friendArr** has the following symmetry property:

Value at entry (*i, j*) = *Value at entry* (*j, i*)

The input to construct the friendship array is as follows:

1. You enter the number of users.
2. You then indicate the number of pairs of direct friends you would like to enter.
3. Lastly, you enter each pair of direct friends.

With this input, your program **friendship.c** will construct the **friendArr** array such that it satisfies the symmetry property.

Subsequently, you are supposed to complete the following two tasks:

a. Write a function `iSolitude()` that displays a list of users (represented by the respective array indices) who have the **LEAST** number of direct friends. For instance, for the small **friendArr** array shown above, `iSolitude()` will print out users 0, 1, 3 and 4 (in ascending order of user ids), as they have the smallest number of direct friends (each one of them has only two direct friends, including himself/herself).

b. The entrepreneur has also requested that you compute the **friend-of-friend** relation, so that if *u* and *v* are direct friends of each other, *iLink* can introduce other direct friends of *u* to *v*, and vice versa. Specifically, *i* and *j* have a **friend-of-friend** relationship if and only if the following two conditions hold:

   i. *j* is NOT a **direct friend** of *i*; and

   ii. There exists a distinct user *k* who is a **direct friend** of both *i* and *j*.

   Write a function **`uFriend()`** that displays all pairs (*i*, *j*) of **friendArr** such that user *i* is a friend-of-friend of user *j*. In the small **friendArr** array shown above, (0, 3) has friend-of-friend relationship, as 0 and 3 are not direct friend of each other, and user 2 is a direct friend of both 0 and 3.

Sample run #1:

```
Read in the number of users: 6
There are 6 users, indexed from 0 to 5.
Enter the number of pairs of direct friends: 5
Enter 5 pairs of direct friends:
0 2
1 5
3 2
4 5
5 1
The friendship matrix is:
  1  0  1  0  0  0
  0  1  0  0  0  1
  1  0  1  1  0  0
  0  0  1  1  0  0
  0  0  0  0  1  1
  0  1  0  0  1  1
The least number of friends found is 2
User 0 has least number of friends
User 1 has least number of friends
User 3 has least number of friends
User 4 has least number of friends
Users (0, 3) have a friend-of-friend relation.
Users (1, 4) have a friend-of-friend relation.
```