2011 Fall EE205

Project #3:
Relationship in a Social Network

Due date: Dec 25th (Sun), 11:59 PM
TA: Donggyu Yun
(dgyun@lanada.kaist.ac.kr)

Ⅰ. Introduction

In this project, you are expected to model practical problems in terms of graphs and come up with solutions. The problem you will be solving is that of query handling in social networks.

In a social network, people are connected to each other by a friend relationship. Your project should take as input the social network and should provide an interface similar to Project #2 to query this social network.

Ⅱ. Input

Your program has to take input from a file. The first n lines of the file list the persons participating in the social network. The next line is a $ sign by itself. Then there are m lines each of which has two persons connected by a friend relationship. A typical input file might look like

```
Omar
Sally
Shantal
Billy
Diego
Prabhu
Natasha
$
Billy Shantal
Diego Sally
Shantal Prabhu
Diego Omar
Natasha Sally
Omar Sally
Diego Natasha
```

The friend relationship is mutual. For example, the line

```
Omar Sally
```

would be equivalent to

```
Sally Omar
```

You can assume that the input has no redundant (equivalent) lines and that the two people in each relationship have been listed before the $. Names do not have white space in them and are *case sensitive*. The two friends on each line of the second part of the file are separated by white space.

Once your program has read in all the names and relationships from the input, it has to accept a series of queries from standard input and generate output to standard output as described for each query below. The series of queries will terminate with a **`quit`** command.

Ⅲ. Queries and Output

It is assumed that your program creates an undirected graph based on the social network and runs algorithms on it in response to each query. The program should begin by printing a $ on a line by itself on the console. After that, the output of each query is terminated with a $ on a line by itself.

Your program should support the following queries.

- **`isfriend <person A> <person B>`**

The output should be a simple yes or no (followed by a $ and a newline). Given the input above, the queries below would result in the output shown

```
isfriend Natasha Diego
yes
$
isfriend Omar Natasha
no
$
```

- **`mutual <person A> <person B>`**

If there are one or more mutual acquaintance(s), you should output all their names one per line, followed by a $ sign on a fresh line.

If there are no mutual acquaintances, you should output the $ sign only. So

```
mutual Prabhu Billy
Shantal
$
mutual Omar Natasha
Diego
Sally
$
mutual Sally Prabu
$
```

- **`relation <person A> <person B>`**

For this query, you should find the shortest sequence of friends between A and B, i.e. shortest path in the graph between nodes representing A and B. Your program should print the names of all the friends on the path in order including A and B, again one name per line. The sequence of friends should be terminated with a $ sign on a fresh line. If there is no path from A to B, then you should output the $ sign only.

```
relation Natasha Omar
Natasha
Sally
Omar
$
```

```
relation Diego Billy
$
relation Diego Natasha
Diego
Natasha
$
```

Ⅳ. Requirements for Implementing the Graph Queries

Your program will read an input text file, respond to a sequence of queries from standard input, and output the results of these queries. It must be executed using the following command:

```
./SocialNetwork  input_file
```

This will be followed by a sequence of query commands coming from standard input (the console). The program responds by writing to the console as shown in the examples above, the output of each query ending with a $ on a line by itself. The program also prints a $ at the beginning to indicate that it is ready to process queries (so that you can tell whether or not it has finished reading the graph).

Instead of typing commands on the console, it may be more convenient to run it as

```
./SocialNetwork input_file < command_file > output_file
```

where output_file will hold the outputs from the commands.

Ⅴ. What To Do

Read ‘chapter 6’ of “Fundamentals of Data structures in C” and implement a graph ADT (see page 271, you can add some functions to the given ADT). You can notice that the textbook handles three kinds of representations for undirected graphs: adjacency matrices, adjacency lists, and adjacency multilists. However, you MUST choose adjacency matrices for implementing a graph ADT.

In this project, your program should support every query mentioned in IV, **`isfriend, mutual, relation,`** and **`quit.`** Your program will be tested by several combinations of queries, for given example input file and TA’s own input file (it will be similar with given one). Followings are an example of expected output result.

```
isfriend Omar Sally
yes
$
relation Natasha Omar
Natasha
Sally
Omar
$
mutual Omar Natasha
Diego
Sally
$
quit
```

Ⅵ. Submission

Submit followings with the title [EE205_project3]20XXXXXX on the moodle page.

(1) Source code

If your source code file name is different with following form, it will be IGNORED. Do NOT separate your code to several files.

(2) Reports

Within 5 pages, do NOT attach WHOLE source code in your report.

1) Explanation or graphical expression about your DATA STRUCTURES.

2) List of FUNCTIONS and their simple explanations (not for all, just for important functions).

3) Explanation or flow chart of your ROUTINEs for handling each query.

Ⅶ. Grading & Delay Penalty

(1) Grading

Grades for various aspects of the assignment	
Compiles and executes without error	10%
Reasonable usages of data structures and readability of source code (easy writing and appropriate comments)	20%
Gives the correct output for commands	50%
Report	20%

(2) Delay penalty

LATE SUBMISSONS ARE NOT ALLOWED.