ECE242_Project 5 Graphs

Friendship Graph

Overview

Most of us use Facebook, one of the most popular online social networks. Facebook offers features of convenience that help users form and maintain an online network with other users. One such a feature is "friend". Via the friend relationships, Facebook users effectively form a graph, called a friendship graph. A friendship graph is a graph where the vertices are users, and there is an edge between two users if and only if they are friends. One conceptual friendship graph is shown in the following figure.

A sample friendship graph from Facebook main page

In this project, using information from the friendship graph, you will implement two strategies for recommending friends to Facebook users: recommend the most popular users among friends' friends, and recommend users who have more than 2 common friends.

Introduction

In this project, you will be given a user file which will be used to construct a friendship graph. You will use the information in the friendship graph to complete the following tasks (each task for one strategy).

1. Recommend the most popular user among a given user's friends' friends

The most popular user is the one who has the most friends. We often make friends with our friends' friends. Hence, we recommend the most popular user among a given user's friends' friends to him/her. That is, for a given user, we find the most popular user among the users who are 2-hop away from him/her, and then recommend the user to him/her.

2. Recommend users who have more than 2 friends in common with a given user.

The second strategy is based on the number of common friends. For a given user, we count the number of common friends he/she and another user have. If this number is larger than 2, that user will be recommended to him/her. In this task, you need to find all users who have more than 2 friends in common with the given user.

Input user file

The input user file includes a set of Facebook users and their friends. Each line describes one user, starts with his/her ID, which is followed by his/her friends' IDs (the first line describes the first user).

For example, the line "1000 1001 1003" indicates user 1000 has two friends, user 1001 and user 1003.

You need to parse the input user file and store the users using either an adjacency matrix or an adjacency list. It is suggested that you use an adjacency matrix to represent the friendship graph to make the coding easier.

Expected classes and methods

1. Implement a class called "**User**" which contains a constructor to initialize the user's ID. You can also define other properties and methods in **User** if you wish.

2. Create a class called "**FriendshipGraph**" which stores the **User**s. Implement the following methods in this class. Feel free to include any additional methods for your convenience.

    a) **parseFile( ) -** This method reads in each line of the user file and stores the users in either an adjacency matrix or an adjacency list. If you use an adjacency matrix to store the graph, you may want to scan the input user file two times. During the first pass, identify the first user listed in each line and store him/her in a user array. An index in the user array can subsequently be used to identify a user. During the second pass, for each line of the input user file, extract the first user and find his/her index in the user array (assume the index is **i**). Then, for each user following the first user (e.g. a user whose index in the user array is **j**), set the value of the adjacency matrix at (**i**, **j**) to 1.

    b) **mostPopularUser(User u)** – This method prints out the ID of the most popular user among user **u**'s friends' friends. For each user in the user array, check whether he/she is 2-hops away from user **u** (if he/she is not **u**'s friend and has at least one common friend with **u**, then he/she is 2-hops away from **u**). If yes, calculate the number of friends he/she has. The user which has the most friends is the most popular user. If there is more than one user who is the most popular, print out the IDs of all of them.

c) **commonFriends(User u)** – This method prints out all the users (it is OK to only print out their IDs) who have more than 2 friends in common with user **u**. For each user who is not **u**'s friend in the user array, compute the number of common friends he/she and user **u** have. If the number is larger than 2, print out his/her ID.

3. In the **main** method, print out the ID of the most popular user among *the first user*'s friends' friends; print out all the users (it is OK to only print out their IDs) who have more than 2 common friends with *the first user*.

4. The above classes and methods can use linked lists, stacks, queues and other data structures. Feel free to add any classes you feel you need to your solution.

Notes

- Make sure your solution is not limited to the sample user file. We may use different user files (with the same format) for grading.

Grading

You should submit your completed code by the date posted on the course website. You will receive 80 points for your code running accurately, and 20 points for the thoroughness and usefulness of your comments which should be included in the code.

- ✓ Parsing the input user file and storing all users correctly (30 points)
- ✓ Finding the most popular user among a given user's friends' friends correctly (25 points)
- ✓ Finding all the users who have more than 2 common friends with a given user correctly (25 points)
- ✓ Commenting thoroughly and usefully (20 points)