In this problem, we need to construct a graph representing the relations between users. In the graph, each vertex represents a user, and each edge represents a relation of direct friends.

Checking with the data quickly, we find there are about 80000 users, but each user averagely has 30 50 friends. In other words, the graph is extremely sparse, and adjacency list representation should be appropriate.

In this python code, the main work is done in the class usernet, which has three attributes:

- bfname: filename of batch_payment info.
- users: a dict storing all users and their friends.
- user_stat: some statistical data of users.

The third one is not necessary for 3 required features.

Aside from constructor, the class has three methods:

- read_batch: read info. from batch_payment.txt and construct the initial state of user network.
- verify: verify if a new payment is trusted.
- update: update the network with new payment info.

1 data structure of user network

The user network is stored in dict users. Each element of users has key as its id, and a set containing all of friends' id's. An example is like below:

users ={'1':{'2', '3'}, '2': {'1'}, '3': {'1'}}

the read_batch method reads "batch_payment.txt" line by line. And for each line, it extracts id1 and id2. It adds id2 to id1's friend and the same is done for id1. If id1(id2) doesn't exist in the network, then the method will create a new key and initialize its friends as {id2(id1)}.

2 verify payment

After user network is created, the code reads "stream_payment.txt" line by line, and calls usernet.verify to test if the payment is trusted.

usernet.verify firstly checks if id1 and id2 both exist in the network. If not, then return False. Then the method tests if id2 is an element of users[id1] (set of friends of id1). This tells if id1 and id2 are "direct friends". If they are not, the method further tests if their friends sets have nonempty intersection, which tells if they have common friends.

If answer is no again, the method continues to test 3rd and 4th degree friendship. For 3rd degree case, the code iterates all friends of id1, and compare

friends set of id1's each friend with id2's friends set, this tells if any friend of id1 has a common friend with id2.

Similarly, for 4th degree case, we iterate all friends of both id1 and id2, and test if anyone of id1's friends has common friend with anyone of id2's friends.

Finally, if all the cases are negative, then the subroutine returns False.

The return value of verify subroutine is a 3-tuple of type (bool, int, int). The first one tells if payment is trusted. The second one tells in which degree the users are "friends" if payment is trusted. The third one tells if payment is "suspicious". (See additional feature)

3 update

After each verification, the main program calls usernet.update to update the user network. It basically does the same thing as read_batch method.

4 additional feature

In addition to the 3 required features, it is instructive to store some statistical information of users and use this information to detect any anomalous behavior.

In this code, I have implemented a simple feature, which involves storing the average value $\bar{X}$ and variance $V(X)$ of payment amount for each user. When a new payment paid by the user is much greater than the user's previous average(I have set the threshold as $\bar{X} + 3 \times V(X)$, roughly corresponding to probability of 99.73%), the verify code will return an alert. This alert is combined with the boolean value showing if payment is trusted. If payment is unverified, then this alert is abandoned, and if payment is trusted but the amount is too large, then the alert signals a suspicious behavior.

5 parameters of main program

The main program accepts parameters for the file names of batch data, stream data and three output files. It accepts one additional parameter "sec_deg" which means security degree. If sec_deg > 1, then the program will switch on the additional feature, and label "suspiciou" payment as "trusted but suspicious".