Programming Language Learning Series
Mastery of Python Language
(Facebook Friend Recommender)

A social network such as Facebook consists of a set of users and connections between the users (i.e., there exists a connection between you and everyone who you have befriended). An interesting problem is to write a computer program that can automatically suggest possible new connections (i.e., friends) for each user.

For this project you will implement a simple system that will, for each user, suggest the most probable user to befriend based upon the intersection of your common friends. In other words, the user that you will suggest to Person A is the person who has the most friends in common with Person A, but who is currently not a friend of Person A.

DataSet: Two files have been provided for you to run your program with. One is a small test file containing a made-up set of users and connections between them (that file is small_network_data.txt). The other is a subset of a real Facebook dataset, which was obtained from: https://snap.stanford.edu/data/egonets-Facebook.html. You can download the files from project directory.

The format of both files is the same: The first line of the file is an integer representing the number of users in the given network. The following lines are of the form: user_u user_v where, user_u and user_v are the IDs of two users who are friends.

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

The above is a representation of a social network that contains 5 users.

Implement the following subtasks:

Build Social Network: The first line of the file contains the value for the variable *n*, the number of users in the social network. In this function, read *n* and build a network using the hints given below.

Implementation Hints:

1. *Initialization:* Create a list containing *n* nested empty lists (e.g., *n* = 3 would correlate to creating the list [[], [], []]). This list of lists will be used to hold the list of friends for each of the *n* users.
2. *Filling up network:* You can then use a for loop to iterate over the remaining lines of the file. For each line of the file you will want to obtain the two user id values

Copyright © Algorithmica
www.algorithmicaonline.com

on that line, we denote those user ids as *u* and *v*. You need to place *v* into *u*'s list and also place *u* into *v*'s list, since they are mutual friends (hint: use the list append() method to place each user id into a list). Finally you will return *network*, which is the list of lists that was created

Number of Common Friends: In this function, you have to calculate the number of common items between two lists.

- *Implementation Hints:* To do this you can have a for loop that iterates over the items of *list1* and then an if statement that checks whether or not that item is also in *list2*. This function is to return an integer, which is the number of items that the two lists had in common (i.e., the number of items that were found in both of the lists). For example, if list1 = [1,3,5,6] and list2 = [1, 2, 3], then the value 2 would be returned since they have two items in common, the values 1 and 3.

Create Similarity Matrix: This function takes the social network as the parameter *network* and creates a "similarity matrix" for all the users in the social network. Here the definition of "similarity" is the number of friends that any pair of users have in common.

Implementation Hints:

1. Initialization: The first thing you will want to do is initialize a list of lists that is *n* x *n* in size to all zeros that will hold the similarity (i.e., number of common friends) for each pairing of users. In other words, you will create a list of lists named *similarity_matrix* that will have *n* nested lists, each containing *n* zeros. (e.g., *n*=3 gives the following list [[0,0,0], [0,0,0], [0,0,0]]).
2. Filling up matrix: You will then use two nested for loops to iterate over all pairs of users in the *network*. For each pair of users you can obtain their list of friends from network, and then call the function Number_of_common_friends, which will return an integer representing the number of friends those two users have in common. The returned value will be stored in locations [user1][user2] and [user2][user1] of the list of lists *similarity_matrix*.

Implementation Hint: You will likely need to store two types of data in two different data structures. In one, you will have a dictionary that stores single words as the key with the value as the set of documents(numbers) that the word appears in. In the other data structure, you will store the actual text of the documents, so that you can display it for the user when they ask. You can use a list or a dictionary for the second data structure.

Copyright © Algorithmica
www.algorithmicaonline.com

Programming Language Learning Series
Mastery of Python Language
(Facebook Friend Recommender)

Friend Recommendations for an user: This function takes *user_id and similarity_matrix* as input and return the most probable friend suggestion for the given user.

Implementation Hints:

1. This *user_id* will be used as an index into the *similarity_matrix* which will give you access to the list of similarity scores for that given user (*user_id*) for all other users. You should then determine the largest value in this list and return its index as the most similar.
2. You don't want to recommend someone who is already a friend and it also doesn't make sense to recommend the person as his or her own friend. It is the most similar because by having the largest value in the list it means that it is the user who had the most friends in common with *user_id*.

Unit Testing: Unit test each function you have written sothat you can be more confident on the project outcome

Main Method: Write the code to integrate subtasks and test it on provided dataset.

Copyright © Algorithmica
www.algorithmicaonline.com