IF100
Files & Dictionary - Practice #4

Introduction

The aim of this example is to practice file operations and dictionaries in Python. The use of file operations is due to the nature of the problem; that is, you cannot finish this practicing assignment without using file operations. You may implement the solution without using a dictionary but that would be less efficient.

Description

In this exercise, you will write a Python program by pretending to be a software engineer in Facebook. We have a real facebook dataset and this dataset contains information regarding the friends of the users. Based on this dataset, you will suggest a number of friends to a specific user depending on a basic algorithm.

Prepared Dataset

The dataset (*`friendship.txt`*) consists of 'friend pairs' from Facebook. Facebook data was collected from survey participants using the Facebook app. For privacy issues, Facebook data has been anonymized by replacing the Facebook-internal IDs with new dummy values.

In each line of the file containing this dataset, there are two integer values and these integer values are separated by the horizontal tab character (\t). Here, each integer value represents the IDs of the users. Hence, each line has the format given below, where *user1* represents the ID of the first user, *user2* represents the ID of the second user and \t is the tab character:

user1\t*user2*

Facebook has an undirected relationship format which means that when you become a friend of a user, then this user also becomes your friend as well. For instance, if you see the line given below in the file (integers are separated by a tab), it means that the user with ID 10 is a friend of the user with ID 158, and it also means that the user with ID 158 is a friend of the user with ID 10.

*10	158*

You may assume that the file is in the correct format for each line and there are no empty lines. However, you cannot make any assumptions on the number of lines and the number of users in the file.

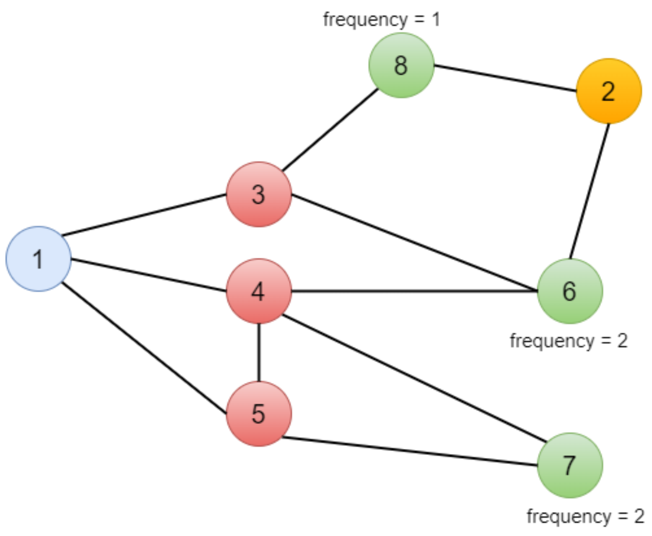
Inputs and Outputs

Your program will prompt for a single input, which is an integer that corresponds to the ID of the user, to whom the friend suggestion will be made. You may assume that this input is really an integer, i.e. it consists only of digits. If this user does not exist in our dataset, i.e. the input user ID does not appear in the prepared dataset file, then your program will prompt "*There is no such user*". Otherwise, your program will suggest (i.e. print) the most frequent user(s) among the friends of the friends of the input user. In other words, it will suggest the most frequent user among the 2nd degree connections of the input user. In Facebook, a friend of you is called as ***your 1st degree connection***. A user is called ***your 2nd degree connection if****:*

- It is not you, and
- It is not your friend, and
- It is a friend of one of your 1st degree connections.

The frequency of a 2nd degree connection is the number of times that the user appears among the friends of the 1st degree connections. Therefore, your program will look for the friends of the 1st degree connections of the input user and find the most frequent user ID within those lists.

If we abstract this problem, we may end up with the following graph, where each user is represented by a node and the friendship information between two users is represented by an edge connecting the nodes of these users:



Here, the blue node (node#1) corresponds to the user to whom the suggestion will be made, the red nodes (node#3, node#4, node#5) correspond to this user's 1st degree connections and the green nodes (node#6, node#7, node#8) correspond to his/her 2nd degree connections. On the other hand, the yellow node (node#2) doesn't have either a 1st degree or 2nd degree connection with the blue node. *frequency* underneath a green node denotes the frequency of the node in consideration with respect to its relation with the blue node. For instance, the frequency of node#6 is *frequency = 2* because of the fact that this node shares 2 common connections (node#3 and node#4) with node#1. Similarly, *frequency = 2* for node#7 because of the fact that this node also shares 2 common connections (node#4 and node#5) with node#1. In such a scenario, node#6 and node#7 will be the nodes to be suggested for node#1 since they have the highest frequency value.

When there are multiple candidates that shares the maximum frequency for friend suggestion, then your program should print each of these candidates in ascending order with respect to their IDs, by separating them with a comma (see sample runs). In case that the input user does not have any 2nd degree friends, then your program should prompt "*There is no friend to suggest*" as the output.

We highly recommend the use of dictionaries.

We are very aware that this homework can also be done by:

1) Using lists instead, which makes the implementation much complicated.

2) Using files only by parsing several times, which is very inefficient.

Also, we would like to remind that this practice example is given for your preparation for the final exam and dictionaries will be an important part of the exam. Doing this practice example by avoiding using dictionaries will not help you much in this direction.

We also highly recommend the use of functions.

Sample Runs

Below, we provide some sample runs of the program that you will develop. The *italic* and **bold** phrases are inputs taken from the user.

Sample Run 1 (*non-existing user*)

```
Enter a user id to suggest some friends: 11
There is no such user
```

Sample Run 2 (*no suggestions*)

```
Enter a user id to suggest some friends: 749
There is no friend to suggest
```

Sample Run 3 (*single suggestion result*)

```
Enter a user id to suggest some friends: 47
324
```

Sample Run 4 (*single suggestion result*)

```
Enter a user id to suggest some friends: 77
280
```

Sample Run 5 (*multiple suggestions result*)

```
Enter a user id to suggest some friends: 294
40, 332
```

Sample Run 6 (*multiple suggestions result*)

```
Enter a user id to suggest some friends: 153
1, 3, 9, 21, 25, 26, 31, 39, 40, 67, 98, 105, 117, 119, 121, 122,
133, 141, 142, 169, 185, 188, 200, 231, 232, 236, 239, 252, 257,
271, 272, 277, 290, 291, 297, 304, 315, 322, 323, 329, 332
```

Programming and Coding Advice

It would be easier for you to implement the algorithm of this problem if you **<u>first</u>** try to **<u>draw the flowchart</u>** or **<u>write the pseudocode</u>** so that you can go over your solution to see if there are any errors.

Additionally, this homework is not short to implement and it will be very difficult and problematic to code the solution without decomposing the process into smaller pieces. Indeed, it is always a good idea to **<u>use decomposition and pattern recognition</u>** in programming, regardless of the length of the problem. Just define subproblems in the whole problem, and try to solve these subproblems before.