CSE 231 **Fall 2022**

Computer Project #8

This assignment focuses on the design, implementation, and testing of a Python program that mostly uses lists and dictionaries to solve some network problems. It also partially uses files, sets and tuples.

It is worth 50 points (5% of course grade) and must be completed no later than **11:59 PM on Monday, November 14, 2022).** After the due date, 1 point will be deducted for every 1 hour late. No more submissions will be accepted after Tuesday November 15, 2022.

If you submit by Sunday, November 13, 2022, you will receive a 2-point bonus points. Note that you need to click on the submit button to be considered as submitted.

Assignment Overview

In this assignment, you will use lists and dictionaries to extract some data from a network of friends. In a social network, every member has some friends, and the friendship relation forms a network (which in mathematics is called a *graph*). Here we deal with some problems such as finding the people with the most friends or the pair of people who are not friends but have the most common friends. This type of analysis can be used beyond everyday social networking to examine criminal and terrorist organizations. Most famously, in 2003, network analysis was used to find Saddam Hussein, President of Iraq, who was hiding from the coalition that invaded Iraq.

Assignment Background

Social networks and the Internet itself are part of the daily life of people. In a social network, one may have some friends—the friendship relationship forms a network of relations. These are well-known with Facebook and Twitter being two well-established social networks. Extracting information from social networks is an important research area of interest to both businesses and industry. In this assignment, we do some simple analysis on a friendship network.

File Specifications

You will develop a Python program that takes two files and extracts some information. The **first file** (`Names.txt`) has the full name of a person in each line. The full names are unique. Assume the line numbers start from zero. The number of the line is the index of the person. In the **second file** (`Friends.csv`) at each corresponding line to the **first file**, there are the indices of the friends.

`Names_small.txt (shorter version of the Names.txt)`

```
Gregory Williams     → name at index 0
Mark Mercado         → name at index 1
Devin Baldwin        …
Samantha Murray
Brandon Garcia
Laura Todd
Alexander Rogers
Michael Walter
Elizabeth Russell
Janet Myers
```

`Friends_small.csv (shorter version of the Friends.csv)`

```
2, 5, 4, 8,          → friend data for name at index 0
0, 6, 7, 6, 2, 3,    → friend data for name at index 1
9, 0, 7, 3, 1,       …
8, 5, 7, 8,
0, 5, 7, 2, 3,
4, 6, 8, 0,
7, 9, 8, 2, 5, 4,
1, 6,
0, 3, 6, 4,
0, 1, 8, 2,
```

Assignment Specifications

You will develop a Python program that has the following functions

`open_file (s) → file pointer:`

a. This function prompts the user to input a file name to open and keeps prompting until a correct name is entered. The parameter, `s`, is a string to incorporate into your prompt so you are prompting the user for a particular type of file (`"names","friends"`).
b. Parameters: `string`
c. Returns: `a file pointer`
d. Display: prompts and error messages

`read_names (fp) → list of strings`

a. This function reads the `Names.txt` file using file pointer, `fp`. The file has no header. There is one name per line—remember to strip off the carriage return. Because order matters you must create the list with the names in the same order that they appear in the file (which basically means simply read it in order). The names are unique (which doesn't matter at first but makes your life easier later when you build a dictionary).
Return the list of names (strings).
b. Parameter: file pointer
c. Returns: list of strings
d. Displays: nothing

`read_friends (fp, names_lst)` → `list of lists of strings`

a. This function reads the `Friends.csv` file using file pointer, `fp`. The file has no header. As shown above (File specification section), each line is a list of indices (ints) of names. That is, line 0, the first line of the file corresponds to the name at index 0 of `names_lst`, and so on.
   However, when you build the list of friends for each name, create a list of names (strings), not a list of indices (but you can create a list of indices first, if you wish, and then convert them to names later). Use the ints in the file to index into `names_lst` to get the corresponding name.
   Return the list of list of names (strings).
b. Parameter: `file pointer, list of strings`
c. Returns: `list of lists of strings`
d. Displays: nothing

`create_friends_dict(names_lst,friends_lst)` → `dict`

a. This function takes the two lists created in the `read_names` function and the `read_friends` function and builds a dictionary. Build a dictionary with name as the key and a list of friends as the value. You can use the fact that the `names_lst` has no repeated names to make this task easier. The lists are parallel in the sense that the friends at index `X` of the `friends_lst` corresponds to the name at index `X` of the `names_lst`. (For those looking for an extra challenge you can do this function in one line using the Python `zip` function.)
b. Parameter: `list of strings, list of lists of strings`
c. Returns: `dictionary` (key is a string; value is a list of strings)
d. Displays: nothing

`find_common_friends(name1, name2, friends_dict)` → `set`

a. This function takes two names (strings) and the `friends_dict` (returned by the `create_friends_dict`) and returns a set of friends that the two names have in common. The names `name1` and `name2` should not be in the set of friends returned. This function is easiest done using Python set operations. Hint: a list can be converted to a set by using the `set()` function, e.g. `set(list).` Once you have friends in sets you can use set operations to get the desired set to return.
b. Parameter: `string, string, dictionary`
c. Returns: `set of strings`
d. Displays: nothing

`find_max_friends(names_lst, friends_lst)` → `list of strings, int`

a. This function takes a list of names and the corresponding list of friends and determines who has the most friends. It returns a list of those with the most friends and how many friends they have. Sort the list of names alphabetically (makes testing consistent). Hint: one approach is to start by building a list of the number of friends that each person has. You can find the `max()` of that list. Then go through that list of the number of

friends and find the indices of those who have `max()` number of friends. (List comprehension is useful here, but not necessary.)

b. Parameter: list of strings, list of list of strings
c. Returns: list of strings, int
d. Displays: nothing

`find_max_common_friends(friends_dict)→ list of tuples, int`

a. This function takes the friends dictionary and finds which pairs of people have the most friends in common. It returns a list of those pairs with the most common friends and how many friends they have. Each pair is represented as a tuple of names (strings). Sort the list of tuples alphabetically (makes testing consistent).
   Note three constraints:
   I. Each pair of people is represented only once. That is, pair (A,B) has a symmetric pair (B,A) which has the same people. Use the pair (A,B) where A < B (the index of A < index of B which happens to be the first one you will consider).
   II. Don't count either A or B as common friends for the pair (A,B).
   III. Don't consider (A,A) as a pair (that "pair" will likely show up as you loop through all possibilities)

   Hint: You will have a nested loop to generate possible pairs. Within that eliminate pairs from consideration based on those three constraints (`continue` is your friend). Use the `find_common_friends` function as it does a lot of your work. After that, finding the max and associated pairs will be similar to your approach in the `find_max_friends` function.
b. Parameter: dictionary
c. Returns: list of tuples of strings, int
d. Displays: nothing

`find_second_friends(friends_dict)→ dictionary`

a. Here we consider second-order friendships, that is, friends of friends. For each person in the network find the friends of their friends, but don't include the person's first order friends or themselves. For example,
   ```
   A: [B,C]
   B: [A,C,D]
   C: [B,E,F]
   ```
   Consider person A who has friends B and C. The friends of friends will be [A,C,D,B,E,F] but we don't count friends B and C, and we don't count A so the second-order friends are [D,E,F], except as noted below you are to return a set. So the entry in the new dictionary for person A will be:
   ```
   A: {D,E,F}
   ```
   Hint: sets work nicely here so create a set of friends of friends, i.e. {A,C,D,B,E,F}, which automatically eliminates duplicates and then use subtraction to remove friends B and C as well as A.
b. Parameter: dictionary
c. Returns: dictionary with key a string and value as a set.
d. Displays: nothing

`find_max_second_friends(seconds_dict)→ list, int`

a. Here we again consider second-order friendships, that is, friends of friends. In the previous function you created a dictionary of such friendships. Now similar to finding max friends you will find max second-order friends. (Note that unlike with friends, here half the work was done in building the `seconds_dict` so this function is short.) Return a list of names (strings) and the int that is the maximum number of second-order friendships.

b. Parameter: dictionary
c. Returns: list of strings, int.
d. Displays: nothing

`main():`

We provide most of `main()`. You only have to handle option 4 where you prompt for a name and print the name's friends. If the name is not a valid name (not in the list of names), re-prompt until it is a valid name.

Assignment Notes and Hints

1. The coding standard for CSE 231 is posted on the course website:

    http://www.cse.msu.edu/~cse231/General/coding.standard.html

Items 1-9 of the Coding Standard will be enforced for this project.

2. The program will produce reasonable and readable output, with appropriate labels for all values displayed.

3. We provide a `proj08.py` program for you to start with.

4. If you "hard code" answers, you will receive a grade of zero for the whole project. An example of hard coding is to simply print the approximate value of e rather than calculating it and then printing the calculated average.

Assignment Deliverable

The deliverable for this assignment is the following file:

`proj08.py` – the source code for your Python program

Be sure to use the specified file name and to submit it for grading before the project deadline.

Function Test
Check the Function_Tests folder

Grading Rubric

Computer Project #08 Scoring Summary
General Requirements:
(4 pts) Coding Standard 1-9
(descriptive comments, function headers, mnemonic identifiers, format, etc...)

Implementation:

(4 pts) open_file function

(4 pts) read_names function

(4 pts) read_friends function

(4 pts) create_friends_dict function

(5 pts) find_common_friends function

(5 pts) find_max_friends function

(5 pts) find_max_common_friends function

(5 pts) find_second_friends function

(4 pts) find_max_second_friends function

(3 pts) Test 1

(3 pts) Test 2 (error in name)

Note: hard coding an answer earns zero points for the whole project (-10) if you do not use main

Test 1:

```
Friend Network

Input a names file: Names.txt
Input a friends file: Friends.csv

 Menu :
    1: Popular people (with the most friends).
    2: Non-friends with the most friends in common.
    3: People with the most second-order friends.
    4: Input memeber name, to print the friends
    5: Quit

Choose an option: 1

The maximum number of friends: 30
People with most friends:
Laura Todd
Sarah White

Choose an option: 2

The maximum number of commmon friends: 10
Pairs of non-friends with the most friends in common:
('Michael Brooks', 'William Chaney')

Choose an option: 3

The maximum number of second-order friends: 169
People with the most second_order friends:
Benjamin Conway

Choose an option: 4
Enter a name: Amanda Barnes

Friends of Amanda Barnes:
Mark Mercado
Miss Kayla Walker
Brandy Vargas
Thomas King
Thomas Perry
Sandra Harris
Larry Powell
George Gomez
Joseph Nguyen
Terry Johns
Tyler Davis
Katie Ferguson
Thomas Bass
```

```
Bryce Garrison
Shawn Logan
Jordan Johnson

Choose an option: 4
Enter a name: Kelly Greer

Friends of Kelly Greer:
Gregory Williams
Devin Baldwin
Christopher Henderson
Jeffrey Glover
David Gomez
Jamie Jackson
Michael Brooks
Ronald Reed
William Chaney
Levi Alexander
Kelly Cunningham
Brady Gomez
Jeremy Burke
Brian Burton
Melissa King
Stephen Parker
James Gonzalez
Richard Horn

Choose an option: 5
```

Test 2:

```
Friend Network

Input a names file: Names.txt
Input a friends file: Friends.csv

 Menu :
    1: Popular people (with the most friends).
    2: Non-friends with the most friends in common.
    3: People with the most second-order friends.
    4: Input memeber name, to print the friends
    5: Quit

Choose an option: 4

Enter a name: Imen Zaabar
The name Imen Zaabar is not in the list.

Enter a name: Amanda Barnes
```

Friends of Amanda Barnes:
Mark Mercado
Miss Kayla Walker
Brandy Vargas
Thomas King
Thomas Perry
Sandra Harris
Larry Powell
George Gomez
Joseph Nguyen
Terry Johns
Tyler Davis
Katie Ferguson
Thomas Bass
Bryce Garrison
Shawn Logan
Jordan Johnson

Choose an option: 5