PROGRAMMING ASSIGNMENT 3

Subject: Graph Theory-Operations on Graph

Teaching Assistant: Burcu YALÇINER

Submission Date: 19.11.2021

Due Date: 03.12.2021 23:00

1. INTRODUCTION

A social network such as Facebook, Twitter, Instagram etc. is a network which is a set of objects (nodes) with interconnections (edges). Here, nodes refer to individuals or things within the network and edges refer to relations/interactions between individuals or things. **Fig.1.** shows a sample social network graph.

Figure 1: Visualization of social network graph

2. ASSIGNMENT

The 3rd assignment is based on this scenario: Suppose that you work for a software company. A new job which is developing a social media platform such as Facebook, Twitter, Instagram etc. is taken by your company. In order to develop a social media platform, you are expected to use a social network described above. Your project manager expects you to manage and investigate the

relation/interaction among the nodes in this social network by using functions, File(I/O), lists, tuples, sets and dictionaries.

At the beginning phase of this part, the file named "**smn.txt**" which will be shared on piazza should be read by your implementation. By using the content of "**smn.txt**", you should create a social network graph. **Fig.2.** shows the content of "**smn.txt**", which consists of adjacency list.

```
Andy:Betty Dacey Hailey Ian Lou Otello
Betty:Andy Dacey Hailey Ian Ken Lou Nadia Ryan Tom
Camila:Emily Otello Ryan
Dacey:Andy Betty Ian Lou Paul Ryan Tom
Emily:Camila Ian John Lou Paul
Frida:Ian Mary Paul Ryan
George:Otello Paul Tom
Hailey:Andy Betty Ian Ken Nadia Ryan
Ian:Andy Betty Dacey Emily Frida Hailey Lou Mary
John:Emily Nadia
Ken:Betty Hailey Nadia Ryan
Lou:Andy Betty Dacey Emily Ian Ryan
Mary:Frida Ian Otello
Nadia:Betty Hailey John Ken Ryan Tom
Otello:Andy Camila George Mary Paul Tom
Paul:Dacey Emily Frida George Otello Ryan
Ryan:Betty Camila Dacey Frida Hailey Ken Lou Nadia Paul
Tom:Betty Dacey George Nadia Otello
```

Figure 2: The content of smn.txt

Fig.3. shows the visualization of social network graph your implementation will create.

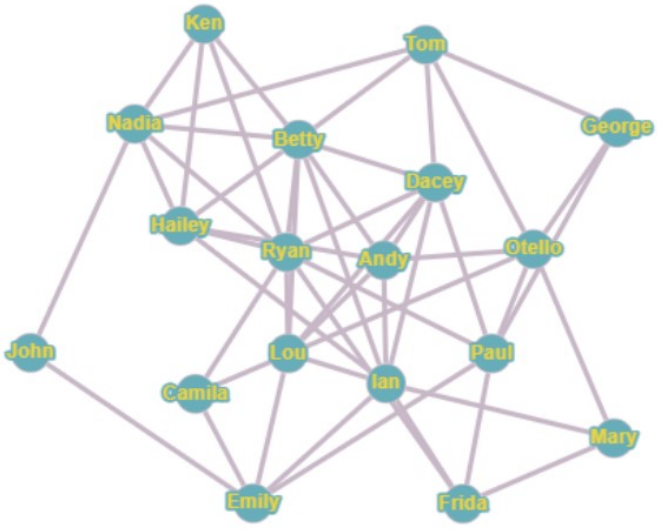

Figure 3: The visualization of social network graph expected

After you construct the social network graph, the file named "**commands.txt**" should be read by your implementation. The details of commands are given in **Table 1**.

Table 1. Commands

COMMAND	TASK
ANU	Add New User
DEU	Delete Existing User
ANF	Add New Friend
DEF	Delete Existing Friend
CF	Give the Number of Existing User
FPF	Find Possible Friends
SF	Suggest Friend to Existing User

Your implementation should perform various functionalities over social network for each command given in **Table 1**. Each of these functionalities is introduced in detail below:

1) **Add New User**: This function should take one parameter which indicates a new user (new node). In this function, new user name is given to a node and it is added into the social network. Command type should be as follows:

ANU < new_user >

There is one restriction:

- When a node is newly created,
  - the new user is not allowed to give an existing name in the social network. i.e., Your implementation should give a message **"ERROR: Wrong input type! for 'ANU'! -- This user already exists!!"**, if the user's name given exists. Otherwise, your implementation should add the user to the social network and give a message **"User <username> has been added to the social network successfully"**.

SAMPLE for Add New User function: When we use the command **ANU<"Sandy">**, the visualization of social network graph will be like:

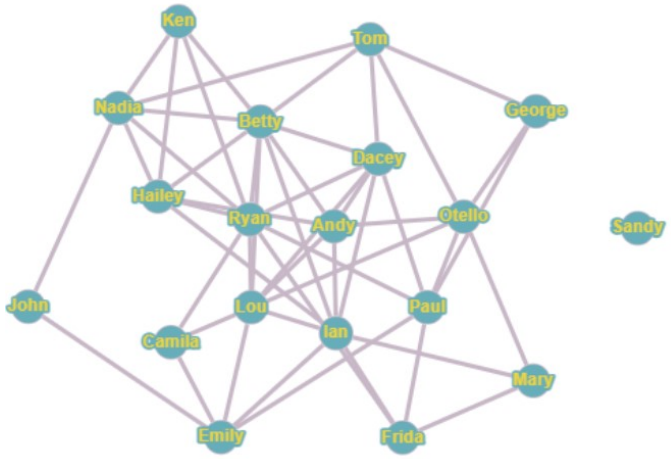

Figure 4: The visualization of social network graph expected after ANU

2) **Delete Existing User:** This function should take one parameter which indicates an existing user (existing node). In this function, the user taken as parameter and his/her all relations should be deleted.
Command type should be as follows:

DEU < username >

There is one restriction:

- A node which does not exist in the social media network is strictly forbidden to delete. i.e., Your implementation should give a message **"ERROR: Wrong input type! for 'DEU'!--There is no user named <username>!!"**, if the user's name given does not exist. Otherwise, your implementation should delete the user from the social network and give a message **"User <username> and his/her all relations have been deleted successfully"**.

SAMPLE for Delete Existing User function: When we use the command **DEU<"Andy">**, Andy and his relations will be deleted. The visualization of social network graph will be like:

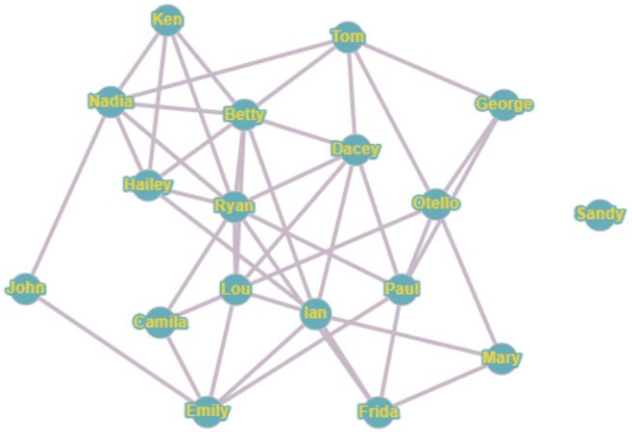

Figure 5: The visualization of social network graph expected after DEU

3) **SAMPLE for Add New Friend function:** This function should take two parameters: first parameter indicates an existing user (existing node) which should be source node and second parameter indicates an existing user (existing node) which should be target node. In this function, a new relation between two nodes is created.
Command type should be as follows:

ANF < source_user > < target_user >

There are two criteria for performing this function:

- Both nodes taken as parameters must exist in social network.

i.e., Your implementation should give either a message **"ERROR: Wrong input type! for 'ANF'! -- No user named <username> found!!"** if one the users given as parameters does not exist in social network or **"ERROR: Wrong input type! for 'ANF'! -- No user named <username> and <username> found!"** if both the users given as parameters do not exist in social network.

- A relation between those nodes must not be created before in social media network.
  i.e., Your implementation should give either a message **"ERROR: A relation between <username> and <username> already exists!!"**.

If the two conditions mentioned above are met, your implementation should add new relation between two nodes and give a message "**Relation between <username> and <username> has been added successfully**".

SAMPLE: When we use the command **ANF<"Otello", "Sandy">**, a relation between Otello and Sandy will be established. The visualization of social network graph will be like:

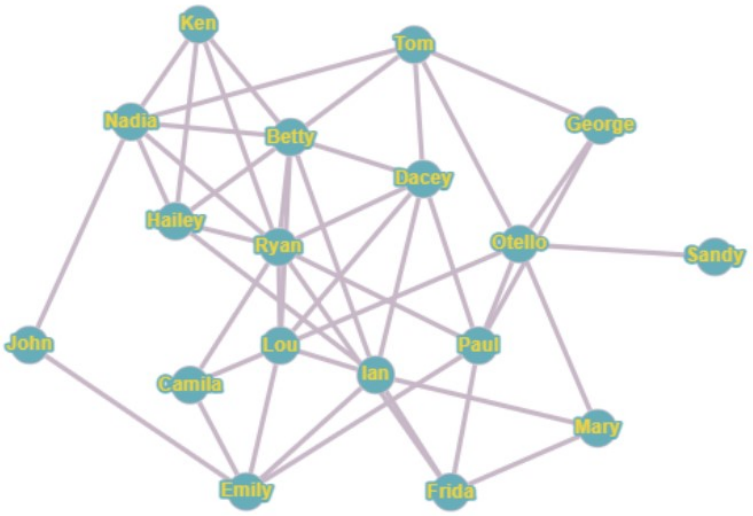

Figure 6: The visualization of social network graph expected after ANF

4) **Delete Existing Friend:** This function should take two parameters: first parameter indicates an existing user (existing node) which should be source node and second parameter indicates an existing user (existing node) which should be target node. In this function, the relation between two nodes is deleted.
Command type should be as follows:

DEF < source_user > < target_user >

There are two criteria for performing this function:

- Both nodes taken as parameters must exist in social media network.

i.e., Your implementation should give either a message

"ERROR: Wrong input type! for 'DEF'! -- No user named <username< found>!" if one the users given as parameters does not exist in social network or
"ERROR: Wrong input type! for 'DEF'! -- No user named <username> and <username> found!" f both the users given as parameters do not exist in social network.

- A relation between two nodes must exist in social media network.
  i.e., Your implementation should give a message **"ERROR: No relation between <username> and <username> found!!"**.

If the two conditions mentioned above are met, your implementation should delete the relation between two nodes and give a message "**Relation between <username> and <username> has been deleted successfully**".

SAMPLE for Delete Existing Friend function: When we use the command **DEF<"Betty", "Dacey">**, a relation between Otello and Sandy will be established. The visualization of social network graph will be like:

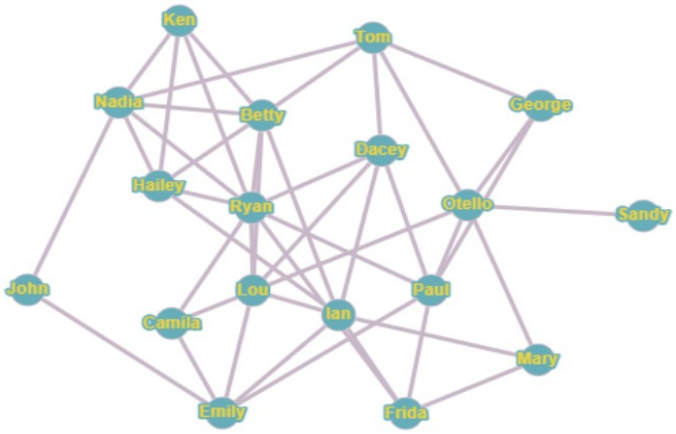

Figure 7: The visualization of social network graph expected after DEF

5) **Count Friend:** This function should take one parameter which indicates an existing user (existing node). This function gives the number of friends of the given user.
Command type should be as follows:

CF < username >

Within the scope of this function, you must avoid to input a non-existing node name. i.e., Your implementation should give a message "**ERROR: Wrong input type! for 'CF'! -- No user named <username> found!"**. Otherwise, it counts the number of friends of the user and gives a message **"User <username> has <friendcount> friends"**.

SAMPLE for Count function: When we use the command **CF<"Tom">**, we will get an output like:

User 'Tom' has 5 friends

6) **Find Possible Friends:** This function should take two parameters: first parameter indicates an existing user (existing node) and second parameter indicates maximum distance. Maximum

distance must be between 1 (included) and 3 (included). (i.e., 1<=maximum_distance<=3) Within the scope of this function, you are expected to find possible friends for each user.
Command type should be as follows:

FPF < username, maximum_distance >

For example, possible friends of "Camila" can be the users shown in list given below when we choose the maximum distance as 2.

{'Andy', 'Betty', 'Dacey', 'George', 'Haley', 'Ian', 'Ken', 'Lou', 'Mary', 'Nadia', 'Otello', 'Paul', 'Ryan, 'Tom'}

Your program should give a message "**ERROR: Wrong input type! for 'FPF'! -- No user named <username> found!"** if the user given as parameter as a non-existing user. Otherwise, it performs its functionality and gives **"User <username> has <count> possible friends when maximum distance is <maximum_distance>"** and **"These possible friends: <possible_friend_list>"**

SAMPLE for Find Possible Friends function: When we use the command **FPF<"Camila",2>**, we will get an output like:

User 'Camila' have 17 possible friends when maximum distance is 2
These possible friends: {'Betty', 'Dacey', 'Emily', 'John', 'George', 'Hailey', 'Frida', 'Ian', 'Ken', 'Lou', 'Mary', 'Nadia', 'Otello', 'Paul', 'Ryan', 'Sandy', 'Tom'}

NOTE: The possible friends for the user given as parameter should be listed in ascending order.

7) **Suggest Friend:** In the last function, you are expected to determine Mutuality Degree (MD) which is a threshold identifying the one as suggested friend for a user (for example node "x"). Also, you are expected to list the possible friends of this user. For example, when we attempt to analyze friendship status for an existing user to suggest friends for him (for example: for node name "Camila" when MD=3. As seen in **Fig. 8**, 'C' has 3 friends {"Emily", "Otello", "Ryan"} and each of them has 5 {"Camila", "Ian", "John", 'Lou"', Paul"}, 6 {"Camila", "George", "Mary", "Paul, "Sandy"," Tom"}, and 9 {"Betty", "Camila, "Dacey", "Frida", "Hailey", "Ken", "Lou", "Nadia", "Paul"} friends, respectively. When considering to the friend lists of Camila 's friends, it can be noticed that only user Paul reaches or exceeds MD, since Camila 's at least 3 friends are friend with Paul. For MD = 2, the suggestion list will become Paul and Lou. Note that, MD must be greater than 1 and user cannot determine a non-existing user or a user having friend less than MD.

Command type should be as follows:

SF < username > < MD >

Your implementation should give a message either
"**Error: Wrong input type! for 'SF'! -- No user named <username> found!!"** if a non-existing user is given as first parameter or "**Error: Mutually Degree cannot be less than 1 or greater than 4"** if either MD is less than 1 or greater than 4. Otherwise, your implementation should give an output as below.

SAMPLE OUTPUT for Suggest Friend function: When we use the command **SF<"Camila",2>**, we will get an output like:

Suggestion List for 'Camila' (when MD is 2):

'Camila' has 2 mutual friends with 'Lou'

'Camila' has 3 mutual friends with 'Paul'

'The suggested friends for 'Camila': 'Lou', 'Paul'

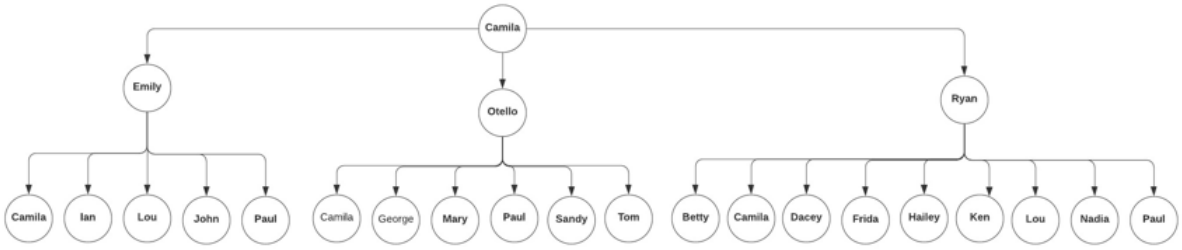

Figure 8: Tree Representation of Friendship

NOTE: The suggested friends for the user given as parameter should be listed in ascending order.

3. SAMPLE OUTPUT

Fig.9. shows the content of a sample “output.txt”.

Figure 9: Sample Output

NOTE 1: Your implementation must process on the file named “**commands.txt**” which will be shared on Piazza.

NOTE 2: Your implementation should give an output file named “**output.txt**” after performing the commands in the file of interest.

NOTE 3: In your assignment, it is expected from you to use File (I/O), dictionaries, lists, tuples and sets in your implementation.

NOTE 4: Your implementation must be original and flexible as much as possible. In other words, your implementation should produce reasonable results when completely different-valid files are used. Otherwise, it will not be considered for evaluation and you will get 0 score from this part.

4. EXECUTION

For execution, the path of the directory where input files exist will be taken from console. To execute your code on dev, firstly use following command line in your terminal:

python3 assignment3.py dirpath dirpath

5. GRADING POLICY

TASK	POINT
SUBMITTED	1
COMPILED	10
OUTPUT	55
DATA STRUCTURES	14
ERROR HANDLING	10
CLEAN CODE	10
TOTAL	100

6. IMPORTANT NOTES

- Do not miss the submission deadline!
- Save all your work until the assignment is graded.
- Your assignment must be original and you must do it with your own individual work. Duplicate or very similar assignments are both going to be considered as cheating. You can ask your questions via Piazza and you are supposed to be aware of everything discussed on Piazza. At Piazza, you cannot share your source code completely or partially. Assignments will be checked for similarity, and there will be serious consequences if plagiarism is detected.
- Your programs will be executed in DEV machine, please make it work on dev before submitting.
- It is your duty to check the Piazza platform against any possible update about this assignment. If any instruction written by the TA violates any condition against this document, the new instruction(s) on Piazza is/are valid!
- You have 3 days extensions for this assignment.
    - 1-day late submission will be graded over 90 points
    - 2-day late submission will be graded over 80 points
    - 3-day late submission will be graded over 70 points
- You will submit your work from department submit system with the file hierarchy as below:
  This file hierarchy must be zipped before submitted (Not .rar, only .zip files are supported by the system)

  →<student id>

  → assignment3.py

- You can lose points if you don't follow the rules defined above.