Management and Analysis of Social Networks

A social network (facebook, twitter, etc.) consists of nodes and edges. In such a network, nodes refer to individuals or things within the network, while edges or *ties* refer to relations or interactions.

Wetherell et al. [1] describe Social Network Analysis (SNA), also called structural analysis, as "*(1) conceptualizes social structure as a network with ties connecting members and channelling resources, (2) focuses on the characteristics of ties rather than on the characteristics of the individual members, and (3) views communities as 'personal communities', that is, as networks of individual relations that people foster, maintain, and use in the course of their daily lives*". In summary, SNA is the process of analyzing the social structure using network or graph theories.

To accomplish this part of the assignment, you are expected to manage and investigate the relation between the nodes in the network, shown in Figure 1, using *File (I/O)*, *dictionaries*, *lists* and *sets*.

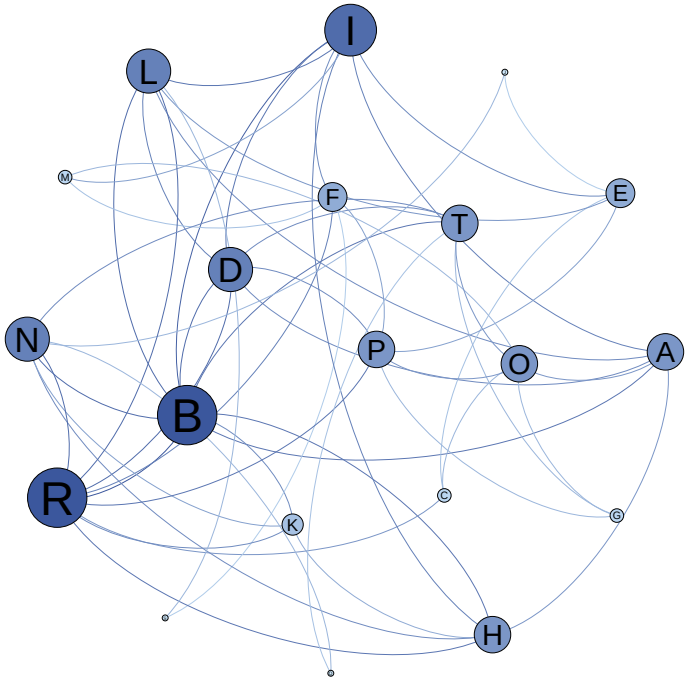

Figure 1: Visualization of a social network graph.

In the beginning phase of this part, your implementation should first read the file "*commandsP1.txt*" which will be shared with you and then perform actions in the direction of those commands[2]. The implementation should have these functionalities over the network:

1. **Add user (`AU`):** In this option, the user gives a name to the node. Note that the user is not allowed to give an existing name to a newly created node. That is why you should check this condition first. The syntax for this command is as follows:

[2]the words and letters in this file are separated by one character of white space (i.e., ' ').

```
AU <username>

examples:
AU      => Missing argument
AU A    => This user already exists!
AU A    => User A added successfully
```

2. **Remove user (`RU`):** In this option, a node whose name is defined by the user and its all relations are removed. The user must be restricted to remove a non-existing node. This process requires only one input, which indicates *existing node*. The syntax for this command is as follows:

```
RU <username>

examples:
RU      => Missing argument
RU A    => There is no user named A
RU A    => User A and its all relations have been removed successfully.
```

3. **Add new relation (`AR`):** A new relation between two nodes is created. Note that the criteria for a new relation are that *i*) there must not exist such a relation created before and *ii*) both nodes must exist in the network. This process requires two inputs: *source node* and *target node*. Because the network graph here is undirected (see Figure 1), do not forget to also add `<source user>` to `<target user>`'s friend list. The syntax for this command is as follows:

```
AR <source user> <target user>

examples:
AR        => Missing argument
AR A      => Missing argument
AR A B    => No user named A or B found!
AR A B    => A relation between A and B already exists!
AR A B    => A relation between A and B has been added successfully.
```

4. **Remove existing relation (`RR`):** An existing relation is broken here. The user must be avoided when attempts to remove a non-existing relation or when inputs a node name that does not exist in the network. As in the previous process, two inputs are required here: *source node* and *target node*. Do not forget to also remove `<source user>` from `<target user>`'s friend list. The syntax for this command is as follows:

```
RR <source user> <target user>

examples:
RR        => Missing argument
RR A      => Missing argument
RR A B    => No user named A or B found!
RR A B    => No relation between A and B found!
RR A B    => A relation between A and B has been removed successfully.
```

5. **Rank users (`PA`):** The user performs degree (or popularity) analysis for each node and displays the top n users according to their popularity. In this assignment, the popularity of node A is equal to the degree of A, that is, the number of friends in A's list. Here, n is determined by the user and must not be greater than the total number of users on the network. This process takes only one input, that is 'n'. The syntax for this command is as follows:

```
PA <toplistsize>

examples:
PA          =>  Missing argument
PA 100      =>  Invalid input since n is greater than X
PA 5        =>
                User 'A' has 7 friends
                User 'C' has 3 friends
                User 'B' has 9 friends
                User 'E' has 5 friends
                User 'D' has 7 friends
                User 'G' has 3 friends
                User 'F' has 4 friends
                User 'I' has 8 friends
                User 'H' has 6 friends
                User 'K' has 4 friends
                User 'J' has 1 friends
                User 'M' has 3 friends
                User 'L' has 6 friends
                User 'O' has 6 friends
                User 'N' has 7 friends
                User 'Q' has 2 friends
                User 'P' has 6 friends
                User 'R' has 9 friends
                User 'T' has 7 friends
                User 'W' has 1 friends
                1. 'B': 9
                2. 'R': 9
                3. 'I': 8
                4. 'A': 7
                5. 'D': 7
```

6. **Suggest friendship (`SA`):** In this command, the user provides Mutuality Degree (MD) as well as a node "i" to list its possible friends. Here, MD is a threshold that is used to suggest one as friend for node i.

   For example, suppose that the user attempts to analyze the status of the friendship to suggest friends to node C when $MD = 3$.

   - As also seen in Figure 2, 'C' has three friends {'E', 'O', 'R'}.
   - Each of them has five {'C', 'I', 'J', 'L', 'P'}, six {'A', 'C', 'G', 'M', 'P', 'T'}, and nine {'B', 'C', 'D', 'F', 'H', 'K', 'L', 'N', 'P'} friends, respectively.
   - When considering the friend lists of C's friends, it can be seen that only node P reaches or exceeds MD, since P is in the friend list of at least three friends of C.
   - For $MD = 2$, the suggestion list will become P and L.

   Note that MD must be greater than one and the user cannot provide a non-existing node or a node having friends less than MD.

```
SA <user name> <MD>

examples:
SA      => Missing argument
SA A    => Missing argument
```

```
SA A 3  => No user named A found!
SA A 3  => User A has friends less than 3
SA A 3  =>
           Suggestion List for 'A' (when MD is 3):
           'A' has 3 mutual friends with B
SA A 2  =>
           Suggestion List for 'A' (when MD is 2):
           'A' has 2 mutual friends with B
           'A' has 3 mutual friends with C
           'A' has 2 mutual friends with D
```

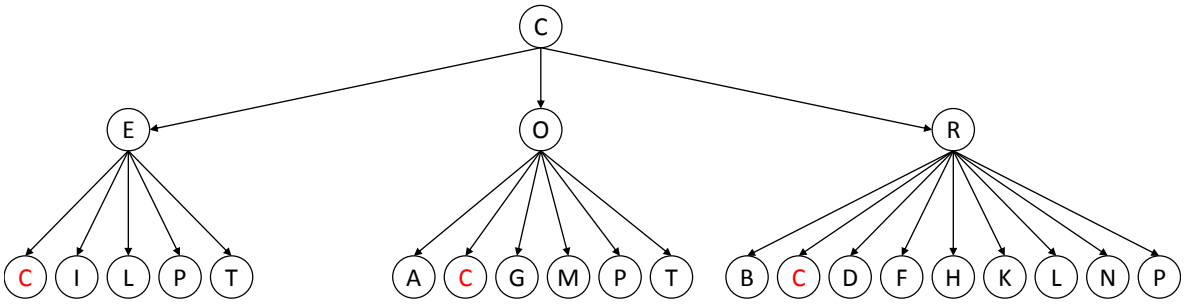

Figure 2: Tree representation of friendship.

Note: After your implementation executes the commands that affect the network graph (i.e., `AU`, `AR`, and etc.), you should write the network with its most recent state into a file named *sn.txt*. In each command step, your implementation should consider the most recent graph.

Command to run this part:

```
python sna.py
```