PROGRAMMING ASSIGNMENT 3

Subject : Graph Operations and Analysis
Advisor : Res. Assist. (Selim YILMAZ, Levent KARACAN, Burçak ASAL, Feyza Nur ÇUBUKÇUOĞLU)

Due Date : 19.11.2015

Part 1: Social network management and SNA

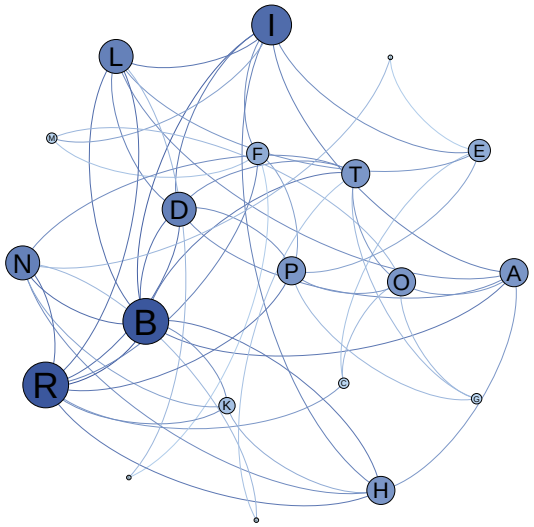

Figure 1: Visualization of social network graph

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 interaction.

Wetherell et al. [1] describes Social Network Analysis (SNA), also called as 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 brief, SNA is the process of analyzing the social structure using network or graph theories.

To accomplish this part of the assignment, it is expected from you to manage and investigate the relation among the nodes in the network, demonstrated in Fig. 1, using *File (I/O)*, *dictionaries*, *lists* and *sets*.

At the beginning phase of this part, your implementation should first read the file "*commandsP1.txt*" which will be shared then produce in the direction of those commands. The implementation should have those functionalities over the network (Fig. 1), each of them is introduced below:

1. **Add user:** In this option, user gives a name to the node and ties it to an existing node. Note that, user is not allowed to give an existing name to a newly created node nor to tie it to non-existing node. This process requires two inputs, first indicates *new node* and latter indicates *existing node*. Command type should be as follows:

$$AU < newuser > < existinguser >$$

2. **Remove user:** In this option, a node whose name is defined by user and its all relations are removed. User must be restricted to remove a non-existing node. This process requires only one input which indicates *existing node*. Command type should be as follows:

$$RU < username >$$

3. **Add new relation:** A new relation between two nodes is created in this option. Note that, the criteria for a new relation are that (1)there must not exist such a relation created before and (2)both nodes must exist in the network. This process requires two inputs, *source node* and *target node*. Command type should be as follows:

$$AR < sourceuser >< targetuser >$$

4. **Remove existing relation:** Within the scope of this option, user determines the nodes' names and must be avoided when he/she attempts to remove a non-existing relation or inputs a non-existing node name. As in the previous process, here two inputs are required, *source node* and *target node*. Command type is given below:

$$RR < sourceuser >< targetuser >$$

5. **Rank users:** User performs degree (popularity) analysis for each node and displays top n users depending on their popularities in this option. Here n is determined by user and must not be greater than total number of user in the network. This process takes only one input "n". Command type should be as follows:

$$PA < toplistsize >$$

6. **Suggest friendship:** In the last option, user determines Mutuality Degree (MD) and a node "i" to list its possible friends. Here, MD is a threshold identifying the one as suggested friend for i. For example, user attempts to analyze friendship status to suggest friends for node C when $MD = 3$. As seen in Fig. 2, *'C'* has 3 friends {*'E', 'O', 'R'*} and each of them has 5 {*'C', 'I', 'J', 'L', P*}, 6 {*'A', 'C', 'G', 'M', 'P', 'T'*}, and 9 {*'B', 'C', 'D', 'F', 'H', 'K', 'L', 'N', 'P'*} friends, respectively. When considering to the friend lists of C's friends, it can be noticed that only node P reaches or exceeds MD, since C's at least 3 friends are friend with P. For $MD = 2$, the suggestion list will become P and L. Note that, MD must be greater than 1 and user cannot determine a non-existing node or a node having friend less than MD.

$$SA < username >< MD >$$

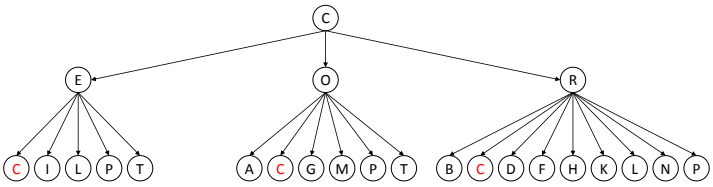

Figure 2: Tree representation of friendship

Note 1: After your implementation performs the commands in the file of interest, you should obtain an output window as in the file named "*outP1.txt*".
Note 2: For the first part, you should operate on the file, (*sn.txt*), which will be shared on *piazza*. At each time you run your implementation, last operated file should be retrieved.

Part 2: Possible path finder

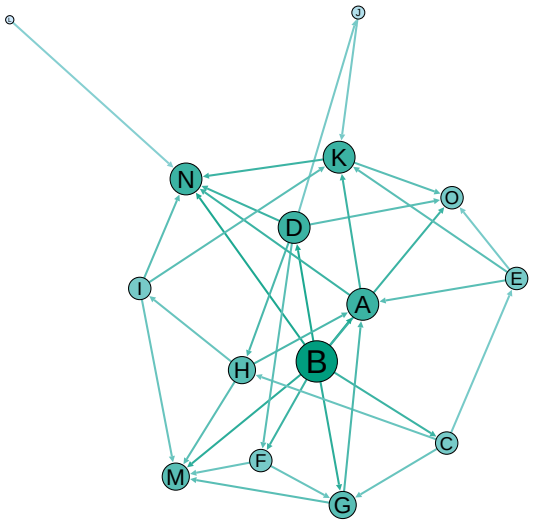

Figure 3: City network

This part of the assignment is based on this scenario: Assume a traveler "*t*" plans to visit some cities in the network given in Fig. 3. As seen in the figure, there is one-way transportation between any of two cities (directions are represented by arrows). Starting from a city, say *x*, *t* can only visit the cities whose maximum distance to *x* is 3. For instance, if *t* starts from *'C'* then the reachable city list is {*'E', 'G', 'H', 'A', 'K', 'N', 'O', 'M', 'I'*}; for *x='J'* reachable city list is {*'K', 'N', 'O'*}.

As in the previous part, your implementation should first read the file, (*commandsP2.txt*), consisting of starting cities then lists reachable cities as given in Fig. 4.

The implementation investigates possible cities based on the descriptions above and it displays the reachable cities as shown in Fig. 4 for the cities given in the input file.

```
'A': ['K', 'N', 'O']
'B': ['C', 'D', 'F', 'G', 'A', 'M', 'N', 'O', 'E', 'K', 'H', 'I', 'J']
'C': ['E', 'G', 'H', 'A', 'K', 'N', 'O', 'M', 'I']
'D': ['F', 'H', 'J', 'N', 'O', 'G', 'A', 'M', 'I', 'K']
'E': ['A', 'K', 'O', 'N']
'F': ['G', 'M', 'A', 'K', 'N', 'O']
'G': ['A', 'M', 'K', 'N', 'O']
'H': ['I', 'A', 'M', 'K', 'N', 'O']
'I': ['K', 'M', 'N', 'O']
'J': ['K', 'N', 'O']
'K': ['N', 'O']
'L': ['N']
City 'M' has no reachable neighbour
City 'N' has no reachable neighbour
City 'O' has no reachable neighbour
```

Figure 4: Output screen

Note 1: As in the previous part, your implementation must process on the file, (*paths.txt*), which will be also shared on *piazza*.

General note: Your implementation must be flexible as much as possible for both parts. Namely, it should produce reasonable results even on completely different network or commands. Otherwise, it will not be considered for evaluation!

General note: Do not manually specify the location of the file, instead use the code block given below:

```
import inspect, os
open(os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe()))) + "\\" + " < file name > "," < mode > ")
```

Notes

- Do not miss the deadline.
- Save all your work until the assignment is graded.
- The assignment must be original, individual work. Duplicate or very similar assignments are both going to be considered as cheating.
- You can ask your questions via Piazza (`https://piazza.com/hacettepe.edu.tr/fall2015/bbm103`) and you are supposed to be aware of everything discussed in Piazza.
- The submissions whose upload score is 0 will not be considered for evaluation.
- You will submit your work from `https://submit.cs.hacettepe.edu.tr/index.php` 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>
→ SNA.py

→ PPs.py
→ commandP1.txt
→ commandP2.txt
→ paths.txt
→ sn.txt

This file hierarchy must be zipped before submitted (Not .rar , only .zip files are supported by the system).

References

[1] Charles Wetherell, Andrejs Plakans, B.W.: Social networks, kinship, and community in eastern europe. The Journal of Interdisciplinary History 24(4), 639–663 (1994)