Algorithms & Data Structures

Week 7 -Graphs

# Problem Introduction

It is often asserted that social network analysis (SNA) can help to identify key players in a network who should be targeted to disrupt organisational terrorist activities. It is thought that measures of centrality might be useful as an investigative tool. The aim of this question is to compute such measures.

We have implemented and provided the class User which represents a user in each network. You **must not modify** this class as it is complete, however you are advised to consult the class documentation before attempting the question.

To achieve our aim, we need to implement the class SocialNetwork representing connection between users. Create a module called socialnetwork where you will implement the class SocialNetwork.

An instance of the class SocialNetwork has the following protected attributes:

* \_name a string containing the name of the Network,
* \_users, a dictionary containing all users from that network mapped by their id.

Note that the class SocialNetwork forms an implicit graph via users' relationships.

## Exercise 1:

Implement the method \_\_init\_\_(name) which takes the name of the Network as a string parameter and initialises the attribute users to an empty dictionary.

## Exercise 2:

Implement the method create\_user(id, name) which creates and returns a new User instance and adds it to the list of users in the social network. The first parameter is the user's id and the second the user's name. The method must raise a ValueError if a user with the same id already exists in this network.

## Exercise 3:

Implement the method get\_user(id) which takes a user's id as parameter and returns the User instance with this id. The method should raise a ValueError if no such user exists.

## Exercise 4:

Implement the method add\_relationship(user\_one\_ID, user\_two\_ID) which adds the user with user\_one\_ID to user\_two\_ID user's connection and vice versa. The method should return True if the connection was successful, False otherwise. The method must raise ValueError if one or both users do not exist. Note, this is the method that creates the edges between the vertices (the users) of the graph.

## Exercise 5:

If you have been using LinkedIn you may have seen that whenever you open your connections, you find 1st, 2nd or 3rd written. It means the following:

* 1st-degree – People you’re directly connected to because you have accepted their invitation to connect, or they have accepted your invitation.
* 2nd-degree – People who are connected to your 1st-degree connections but are not 1sr degree connections.
* 3rd-degree – People who are connected to your 2nd-degree connections, but are not 1st or 2nd degree connection
* Out of Network – LinkedIn members who fall outside of the categories listed above.

Implement the method connexion\_degree(source\_id, target\_id) which takes a source user id, and a target user id, and returns in which of the four categories described above the target user is. The following encoding is used for the return value:

* -1 means “out of network”,
* 1 means “1st-degree”,
* 2 means “2nd-degree”,
* 3 means “3rd-degree”.

The method MUST NOT raise an error if one of the ids does not exist, it should return -1 instead.

## Exercise 6:

Following on exercise 5, implement a method get\_close\_network(user\_id) that returns the set of users’ ids that have a connexion degree of 1, 2, or 3. The method returns an empty set if there is no users with an id equal to user\_id.

## Exercise 7:

There are several centrality measures. The normalised closeness centrality (or closeness) of a node is the average length of the shortest path between the node and all other nodes in the graph. Thus, the more central a node is, the closer it is to all other nodes. Closeness was defined by Bavelas in 1950, and its normalised form is given by Equation 1.

where is the closeness of vertex , is the shortest distance between vertices and , and is the number of vertices in the graph. It is assumed that the graph is strongly connected, that is every vertex is reachable from every other vertex.

Implement the method closeness(user\_id) which computes and returns the closeness measure of the user with ID user\_id. The method should raise a ValueError| if such a user does not exist.