Project report

Introduction:

The project was to implement a graph data structure which represent relationships between people. Each vertex represents a person and is represented by a Person object.

My objective was to create a structure which I can easily use, and access with precise information.

Methodology:

* I started by create the structure for a person, and every function I needed to fully implement and use this structure. At first, I wanted to create a structure list for the friends, but it was complicated for nothing, so I simply used a vector of int.
* Then I implemented the structure for the graph and tested it with only 3 persons manually added.
* When I saw that everything was working, I used the file with all the persons, create the main and added one by one all the functionalities.

Implementation:

* When I create the structure of a person, I created several functions to modify manually all the parameter of a person. Even if I never use it at the end of the project, it helped me at the beginning to make some tests. And they can be useful if later I want to improve this project by giving the opportunity to add and modify a specific person in the graph.
* The functions that suggest friends of a given person:
  + By common friend: It was the most difficult of those functions. First, I get all the friends of the given person. And for each friend I take the id of all his friends. Then I sort the list of id using std::sort from the algorithmfwd.h library. After that I erase all the duplicate with erase from stl\_vector.h and unique from algorithmfwd.h. Now I have a sorted list of possible friends without duplicate, but there is still the possibility that in this list the given person himself, so I erase his id in the list. And then return the list.
  + By occupation: First I check in all the graph if there is someone with the same occupation as the given person, if that is the case, I add his id to the list “sameoccupation”, then I check if those persons are not already friends with the given person, if not I add them to the list “suggestedFriends” and return the list. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
  + By age: At the beginning of the function, I created a int variable named average, it is used to choose by how many years difference you want to suggest friends. Then I go through all the graph and take the id of all the person which age is close enough from the given person. After that I erase the friends that the given person already has and return the list.
* The degree centrality function: To find the degree centrality function, I used the function size from stl\_vector.h to get how many friends each person has.
* The clustering coefficient function: For the clustering coefficient function I used the formula C= number of closed triangles/ number of connected triangles. First of all, I calculated the number of closed triangles. To do that, for each friends of the given person, I get all his friends, and for each of the friends of friends, if this friend is also friend with the given person we add one to the closedtriplet variable and delete it from friends so we won’t count the triangle two times. For the number of connected triangles, I used the formula k(k-1)/2 with k the number of friends of the given person.
* The girvan Newman function: For this function, we will begin by finding the edge with the biggest weight and remove this edge. We will do this step as many times as the user choose when I write the number of iterations. Then we will find the minimum and maximum person ids in the graph. For each person in the graph we will go through the communities and search if this person is already in a community, if that is the case we will add his friends to the community, if not we will create a new community and add his friends. At the end we will erase the duplicate in communities.

Results:

At the end my code work, but I think it is a little bit too complex and I could have done easier for some part.

Here some screenshots of the result.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Here the result of the main, with the selection

Une image contenant texte, capture d’écran, Police, menu

Description générée automatiquement

This is how I display the different person in the graph.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

One example of the suggestion of friends by common friends

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

How I display the degree centrality

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

The display of the clustering coefficient

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

The function Girvan-Newman for one iteration

Conclusion:

This project made me manipulate vectors and different types of structures and helped me gain a deeper understanding of some basic c++ knowledge.

There is still some problem:

* If I want a big number of iterations in the Girvan-Newman function, the different community are strange.
* In the function suggest friend by similar occupation, my code can not detect the similarity if the occupation is not exactly the same.
* The display in the screen could be improve.

But even if I still have some problem I think mostly answer the goal of the project which was to provide possibility to manipulate and get information about a social network.

To change : id 2 times

si pas exactement meme mot dans occupation va pas les associer