Feuille de TP No 1
Advanced Algorithms
Blerina Sinaimeri

Mars 2017

1 Première partie

Cherchez sur internet des graphes réel que vous pouvez facilement convertir dans un format manipulable sur votre machine (vous garderez l'adresse de des données en commentaire dans votre fichier). Variez les formats d'origine si vous pouvez !

Exercice 1. Pour chaque graphe réel récupéré, décrivez vos données. En particulier,

- Tracez la distribution empirique des degrés du graphe. Calculez le degré moyen, comparez au nombre d'arêtes.
- Connaissez-vous le nombre de composantes connexes dans votre graphe ?

Solution 1. You can find a lot of real life graphs in internet or you can even create your own from real life. However, it may be useful in the future to know some nice websites that contain datasets. For example two good ones are `https://icon.colorado.edu/#!/networks` and `http://www-personal.umich.edu/~mejn/netdata/`.

For this exercise I choose to represent the graph representing the metro lines of Lyon. The graph is shown in Fig.1. This graph has 11 vertices.

- For the graph depicted in Fig.1 we have 11 vertices and the empirical distribution of degrees $f_1 = 6/11, f_2 = 1/11, f_3 = 2/11, f_4 = 2/11$. The average degree is $22/11 = 2$ and the number of edges is 10.
- The graph is connected and hence has only one connected component.

Exercice 2.

- Écrivez une fonction qui prend en entrée une matrice d'adjacence et donne en sortie une liste de sommets accompagnés de la liste de leurs voisins.

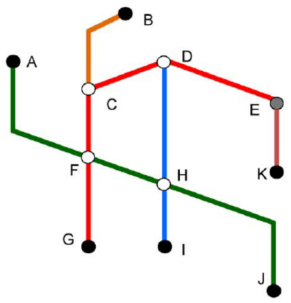

FIGURE 1 – The graph representing the lines of metro in Lyon with their intersection points and end points as vertices.

— Écrivez une fonction pour lire une liste comme ci-dessus et fabriquer la matrice d'adjacence correspondante.

Solution 2. I am providing the pseudocode in python style of the first case. The second is similar.

```
def matrix2list(matrix)
adjList = list()
rows,columns =matrix.shape
For  in range(0, rows-1):
        For j in range(i+1, columns-1)
           if matrix[i,j]=1 :
               adjList[i].append(j)
return adjList
```

Exercice 3. Écrivez une fonction qui prend en entrée un graphe (une matrice d'adjacence ou une liste d'adjacence) et donne en sortie la distribution des degrés du graphe. Vous préférez utiliser une matrice d'adjacence ou une liste d'adjacence ?

Solution 3. Here you could use an adjacency list.

Exercice 4.

— Écrivez une fonction qui prend en entrée un graphe (une matrice d'adjacence ou une liste d'adjacence) et un sommet v et donne en sortie le *individual clustering coefficient* de v.
— Écrivez une fonction qui prend en entrée un graphe (une matrice d'adjacence ou une liste d'adjacence) et donne en sortie le *overall clustering coeffiecient* de G.

Solution 4.

```
#we suppose G is given as an adjacency matrix
#and i is an index in this matrix that corresponds to the desired vertex.
def individual_clustering(i, matrix_G)
individual_clustering=0
d=0
rows,columns =matrix_G.shape
For j in range(i+1, columns-1) :
     if matrix[i,j]=1: d=d+1
    For k in range(j+1, columns-1):
#chek if j and k are both adjacent elements of i and are both connected.
#Thus, we count one triangle
        if ((matrix[i,j]=1) and (matrix[i,k]=1)  and (matrix[j,k]=1)):
                  nr_triangles=nr_triangles+1
individual_clustering=nr_triangles/((1/2)d(d-1)
return individual_clustering
```

For the next part we could use the function we already defined.

```
def overall_clustering(matrix_G)
overall_clustering=0
rows,columns =matrix_G.shape
For i in range(0, rows-1) :
        overall_clustering=overall_clustering+ individual_clustering(i, matrix_G)
return individual_clustering/rows
```

Exercice 5. * Écrivez une fonction qui prend en entrée un graphe $G = (V, E)$ (une matrice d'adjacence ou une liste d'adjacence) d'́etermine si G contient des cycles de longueur 4. L'algorithme doit avoir la complexité $O(n^2)$.

Solution 5. I will give only a hint for this exercise. Observe first that a naive approach would check every 4 vertices and see if they form a cycle. This will require $O(n^4)$ time. The trick to reach the desired time complexity is to keep an additional matrix $C_{n\times n}$. Initially $\forall 0 \leq i, j \leq n$, $C[i, j] = 0$. Then consider a vertex x and let u, v be two of its adjacents. Then u and v are connected by an edge then add 1 to $C[u, v]$ (meaning that there is a path from u to v of length 2 that passes through x). The first time that $C[i, j]$ is at least 2 we have a 4 cycle. This is because having $C[i, j] \geq 2$ means that there are at least two different paths between u and v of length two, hence this will form a C_4. Now for the complexity of this algorithm, observe that for every three vertices x, u, v we either set $C[u, v]$ to 1 or we stop. Hence, every cell of the matrix C will be seen exactly once before we finish. We assume the graph is in adjacency list representation, so it is easy to select pairs of neighbors of a vertex.

2 Deuxiéme partie

Dans ce TP, on utilisera le package igraph. Nous devons commencer par charger le package igraph. Et si cest la première fois, il va aussi falloir les installer. L' installation se fait en tapant

$$> install.packages(\text{"}igraph\text{"})$$

On doit maintenant le charger, ceci se fait avec la fonction suivante (à chaque utilisation de R, le faire une fois au dámarrage et ensuite cest bon) :

$$> library(igraph)$$

Exercice 6. En utilisant le package igraph :

- Créer un graphe complet à 10 sommets.
- Créer un graphe etoile à 10 sommets.
- Comment savoir si deux somments i et j son connectès ?
- Créer un graphe à partir d'une matrice d'adjacence et visualisez le en utilisant $plot()$.

Exercice 7. Familiarisez vous avec les fonctions de conversion des graphes dans les differents formats, avec les fonctions de visualisation et les statistiques descriptives disponibles. Reprenez vos exemples ci-dessus.

Solution 7. To answer the questions in this part, have a look at this tutorial on the package igraph `http://www.dil.univ-mrs.fr/~tichit/rb/tp1/igraph_tutorial.html` .