TP-Théorie des graphes et Algorithmie(Python)

Master 1 OIVM-UPEC

Remarques !!

Il est demandé aux étudiants de rendre un compte rendu du TP à la fin de chaque semaine (au plus tard samedi à 23H59)

Installation de l'environnement du travail

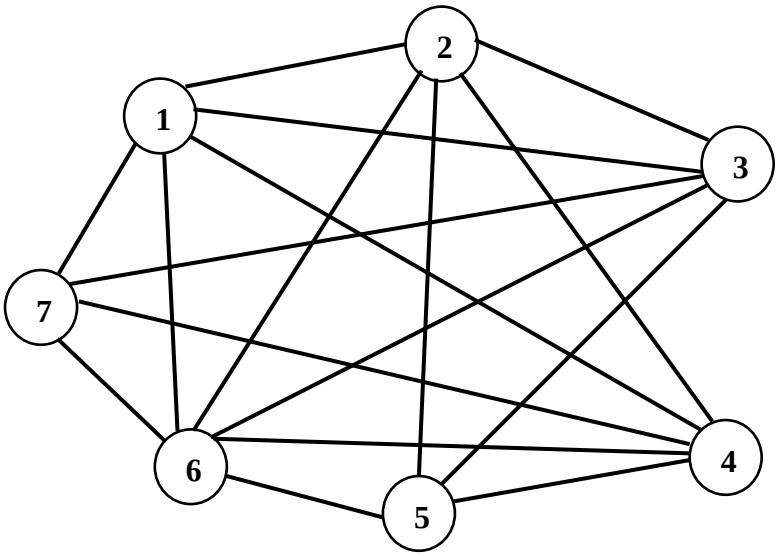
Avant de commencer le TP il faut installer :

- Installation Python sur Windows: https://www.python.org/ftp/python/3.7.9/python-3.7.9.exe
- Installation Anaconda sur Windows: https://docs.anaconda.com/anaconda/install/windows/

Le lien suivant décrit l'installation de l'environnement Python avec Anaconda : https://mrmint.fr/installer-environnement-python-machine-learning-anaconda

Exercice1

Considérons un graphe non orienté $G = (V, E)$ qui est donc simplement constitué d'une liste de n sommets et m arêtes reliant deux des sommets du graphe. Un exemple d'un tel graphe :



Un tel graphe sera représenté dans un premier temps en Python par la liste de ses arêtes, chaque arête étant elle-même définie par une liste de deux nombres entiers contenant les numéros des deux sommets reliés par l'arête. Ainsi, le graphe précédent sera représenté par la liste suivante : $G = [[1,2],[1,3],[1,4],[1,6],[1,7],[2,3],[2,4],[2,5],[2,6] \ldots\ldots]$.

1. Écrire une fonction **taille(listeGraphe)** qui revoie la taille du graphe G représenté par la liste **listeGraphe**.
2. Écrire une fonction **ordre(listeGraphe**) qui retourne l'ordre du graphe G.
3. Écrire une fonction **degre**(**listeGraphe, n**) qui renvoie le nombre de voisins du sommet n dans le graphe G.
4. Écrire une fonction **voisins(listeGraphe, n**) qui renvoie la liste des voisins du sommet n dans le graphe G.
5. Écrire une fonction **connexe(listeGraphe**) qui déterminer si le graphe G est connexe(la fonction renverra True si c'est le cas, False sinon).
6. . Écrire une fonction **complet(listeGraphe**) qui déterminer si le graphe G est complet(la fonction renverra True si c'est le cas, False sinon).
7. Écrire une fonction **regulier**(l**isteGraphe**) qui déterminer si le graphe G est régulier, c'est-à-dire si tous ses sommets ont le même degré (la fonction renverra True si c'est le cas, False sinon).

Exercice2

Dans cet exercice nous utiliserons deux méthodes possibles pour représenter un graphe orienté :

- Une matrice d'adjacence M telle que $M[x][y]$ vaut 1 s'il y a un arc de x vers y, ou 0 sinon. En Python ce sera une liste de listes.
- Une liste d'adjacence pour chaque sommet. En Python on pourra utiliser un dictionnaire dont les clés sont les sommets, et la valeur associée à un sommet est la liste de ses successeurs.

1. Dessiner le graphe dont la représentation par listes d'adjacence est fournie.
2. Écrire une fonction **mat_adjacence(listeAdjGraphe)** qui prend en argument un dictionnaire de listes d'adjacence et renvoie la matrice du graphe correspondant.
3. Écrire la fonction réciproque **listes_adjacence(matAdjGraphe)** qui prend en argument une matrice d'adjacence et renvoie le dictionnaire de listes d'adjacence correspondant.

4. Vérifier sur le graphe fourni en exemple que les deux transformations sont bien réciproques.

Exercice3

1. Écrire une fonction **nonoriente(M)** qui prend comme argument une matrice carrée M (qu'on suppose remplie uniquement de 0 et de 1), et qui détermine s'il s'agit de la matrice d'un graphe non orienté (ce sera le cas si chacune des arêtes peut être parcourue dans les deux sens, donc si $M[i][j]$ est toujours égal à $M[j][i]$).
2. Écrire une fonction **distance(M,i,j)** qui calcule la distance entre les sommets i et j dans le graphe dont la matrice d'adjacence est M, puis une fonction **diametre(M)** qui calcule le diamètre du graphe de matrice d'adjacence M .

Exercice3 (Modélisation)

Un passeur se trouve au bord d'une rivière avec un loup, une chèvre et une salade. Comme vous le savez probablement, les loups mangent les chèvres mais pas les salades, les chèvres mangent les salades mais pas les loups, et les salades ne mangent personne. Dans sa barque, le passeur ne peut transporter qu'un seul des trois protagonistes à la fois. Lorsqu'il est dans sa barque ou sur la rive opposée, il ne peut empêcher de carnage alimentaire.

On souhaite savoir s'il peut amener, sains et saufs, de l'autre côté de la rive le loup, la chèvre et la salade. Si cela est possible, combien de traversées sont nécessaires ?

Pour répondre à ce problème, vous en donnerez une modélisation par un graphe, et le reformulerez dans ce cadre.