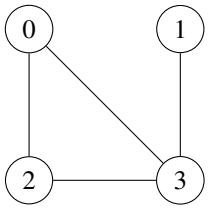
TP 12 : REPRÉSENTATIONS DES GRAPHES

L'objectif de ce TP est de manipuler les différentes façon de représenter un graphe. On ne s'intéresse aujourd'hui qu'aux graphes non orientés, non pondérés.
Dans les exemples suivants on considère ce graphe non orienté :



Rappels : on a vu dans le cours trois façons d'implémenter un graphe en python :

1. À l'aide d'une matrice d'adjacence représentée par une liste de listes :

```
G = [[0, 0, 1, 1], [0, 0, 0, 1], [1, 0, 0, 1], [1, 1, 1, 0]]
```

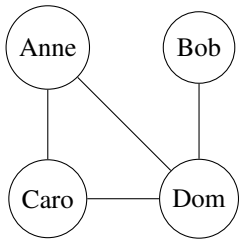
   représentant ici la matrice :

$$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

2. Par listes d'adjacence, implémentée par une liste de listes : chaque élément de la liste est une liste de voisins :

```
G = [[2,3], [3], [0, 3],  [0, 1, 2]]
```

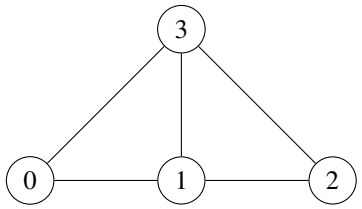
3. Lorsque les sommets ne sont pas étiquetés de 0 à $n-1$, il peut être plus opportun de représenter le graphe par un dictionnaire des listes d'adjacence, où les clés sont les sommets et les valeurs sont les listes de voisins. Par exemple :



   serait représenté par :

```
G = {"Anne" : ["Dom", "Caro"], "Bob" : ["Dom"], "Caro" : ["Anne", "Dom"],
"Dom" : ["Anne", "Caro", "Bob"]}
```

Pour les exercices 1 et 2, on donne le graphe non orienté $\mathscr{G}$ suivant :



Exercice 1 (Représentation par la liste des listes d'adjacence)**.**

1. Écrire la liste `G1` représentant le graphe $\mathscr{G}$ par liste de listes d'adjacence.
2. Écrire une procédure `ajoute_sommet1` qui prend en paramètre un graphe `G` représenté par une liste des listes d'adjacence et ajoute un sommet isolé.
3. Écrire une procédure `ajoute_arete1` qui prend en paramètre un graphe `G` représenté par une liste des listes d'adjacence et une arête `a = [s1, s2]` et ajoute l'arête au graphe. Les sommets `s1` et `s2` sont supposés appartenir au graphe.
4. Écrire une fonction `ordre1` qui prend en paramètre un graphe `G` représenté par une liste des listes d'adjacence et qui renvoie le nombre de sommets du graphe.
5. Écrire une fonction `degre1` qui prend en paramètre un graphe `G` représenté par une liste des listes d'adjacence et un sommet `s` et qui renvoie le nombre de voisins de `s`.
6. Écrire une fonction `sont_voisins1` prenant en paramètre un graphe `G` représenté par une liste des listes d'adjacence et deux sommets `s1` et `s2`, et testant si ces sommets sont voisins dans le graphe.
7. Écrire une fonction `nb_aretes1` qui prend en paramètre un graphe `M` représenté par une liste des listes d'adjacence et qui renvoie le nombre d'arêtes du graphe.
8. Écrire une fonction `regulier1` qui prend en paramètre un graphe `G` représenté par une liste des listes d'adjacence et qui renvoie `True` si le graphe est régulier, c'est-à-dire si tous les sommets ont le même nombre de voisins et `False` sinon.
9. Déterminer la complexité des fonctions précédentes, en fonction du nombre de sommets n du graphe.

Exercice 2 (Représentation par une matrice d'adjacence)**.**

1. Écrire la liste `M` représentant le graphe $\mathscr{G}$ par une matrice d'adjacence.
2. Écrire une procédure `ajoute_sommet2` qui prend en paramètre un graphe `M` représenté par une matrice d'adjacence et qui complète la matrice pour ajouter au graphe un sommet isolé.
3. Écrire une procédure `ajoute_arete2` qui prend en paramètre un graphe `M` représenté par une matrice d'adjacence et une arête `a = [s1, s2]` et ajoute l'arrête au graphe. Les sommets `s1` et `s2` sont supposés appartenir au graphe.

4. Écrire une fonction `ordre2` qui prend en paramètre un graphe `G` représenté par une matrice d'adjacence et qui renvoie le nombre de sommets du graphe.
5. Écrire une fonction `degre2` qui prend en paramètre un graphe `M` représenté par une matrice d'adjacence et un sommet `s` et qui renvoie le nombre de voisins de `s`.
6. Écrire une fonction `sont_voisins2` prenant en paramètre un graphe `G` représenté par une matrice d'adjacence et deux sommets `s1` et `s2`, et testant si ces sommets sont voisins dans le graphe.
7. Écrire une fonction `nb_aretes2` qui prend en paramètre un graphe `M` représenté par une matrice d'adjacence et qui renvoie le nombre d'arêtes du graphe.
8. Écrire une fonction `regulier2` qui prend en paramètre un graphe `M` représenté par une matrice d'adjacence et qui renvoie `True` si le graphe est régulier, c'est-à-dire si tous les sommets ont le même nombre de voisins et `False` sinon.
9. Écrire une fonction `aretes2` qui prend en paramètres un graphe `M` représenté par sa matrice d'adjacence et qui renvoie la liste des arêtes.
10. Déterminer la complexité des fonctions précédentes, en fonction du nombre de sommets n du graphe.

Exercice 3 (Conversions)**.**

1. Écrire la fonction `conversion1` qui prend un graphe implémenté par une liste des listes d'adjacence et qui renvoie la matrice d'adjacence correspondante.
2. Écrire la fonction inverse `conversion2` qui prend en paramètre une matrice d'adjacence et renvoie la liste des listes d'adjacence correspondante.

Exercice 4 (Nombre de Erdös)**.** Le nombre d'Erdös d'une personne est sa « distance de collaboration » avec le mathématicien hongrois Paul Erdös (1913-1996), mesurée par publication conjointe.
Si une personne a un nombre de Erdös qui vaut 1, cela signifie qu'elle a écrit un article de recherche avec Erdös ; un nombre de Erdös égal à 2 signifie qu'elle a cosigné un article avec un collaborateur direct d'Erdös mais pas avec Erdös lui-même, etc. Plus formellement, le nombre d'Erdös d'une personne peut être défini par récurrence de la façon suivante : le nombre d'Erdös de Paul Erdös vaut zéro, le nombre d'Erdös d'une personne M est le plus petit nombre d'Erdös de toutes les personnes avec qui M a cosigné un article, plus un, si M n'est pas collaborateur direct ou indirect d'Erdös, son nombre d'Erdös est infini.
Il est possible de trouver son nombre de Erdös à l'adresse :
`https://mathscinet.ams.org/mathscinet/collaborationDistance.html`
Par exemple Devys,A a un nombre de Erdös de 4.

Supposons qu'on ait un groupe de chercheurs publiant en mathématiques. Voici les interactions que l'on a recensées :

- André a co-écrit un article avec Béa, Charles, Estelle et Fabrice ;
- Béa a co-écrit un article avec André, Charles, Denise et Héloïse ;
- Charles a co-écrit un article avec André, Béa, Denise, Estelle, Fabrice et Gilbert ;
- Denise a co-écrit un article avec Béa, Charles et Estelle ;
- Estelle a co-écrit un article avec André, Charles et Denise ;
- Fabrice a co-écrit un article avec André, Charles et Gilbert ;
- Gilbert a co-écrit un article avec Charles et Fabrice ;
- Héloïse a co-écrit un article avec Béa.

1. Faire le schéma du graphe correspondant à ce groupe.
2. On décide de représenter ce réseau en Python par un dictionnaire de listes d'adjacence. Écrire la définition du dictionnaire `G3` représentant ce graphe.
3. Écrire une fonction `conversion3` qui prend en argument un graphe représenté par un dictionnaire des listes d'adjacence, et qui renvoie la matrice d'adjacence et un dictionnaire dont les clé seront le noms des chercheurs et les valeurs associées sont l'indice du sommet dans la matrice.
4. Écrire une fonction `produit` qui calcule le produit de deux matrices carrées de même taille. Une matrice sera représentée par la liste des lignes de la matrice.
5. Soit `M` la matrice d'adjacence associée à notre réseau social. En testant sur différentes valeurs, conjecturer à quoi correspond le coefficient à la i^e ligne et la j^e colonne de la matrice M^k.
6. La distance entre deux sommets i et j d'un graphe est le nombre minimal d'arêtes que l'on doit parcourir pour atteindre le sommet j depuis le sommet i. Le diamètre d'un graphe est la distance maximale entre deux sommets de ce graphe (on suppose ici que nos graphes sont connexes, c'est-à-dire qu'on peut toujours atteindre un sommet du graphe en partant d'un autre quelconque de ses sommets).
   En exploitant le résultat précédent, écrire une fonction `distance` qui prend en argument la matrice d'adjacence, et deux indices `i` et `j` et qui renvoie la distance entre ces deux sommets, puis `diametre` qui prend en argument une matrice d'adjacence et qui renvoie le diamètre du graphe qu'elle représente.
7. En déduire une fonction `distance_collab` qui prend en argument un graphe représenté par un dictionnaire des listes d'adjacences et les noms de deux chercheurs et qui renvoie la distance de collaboration entre ces deux chercheurs. Écrire également `diametre_collab` qui prend en argument un graphe représenté par un dictionnaire des listes d'adjacences et renvoie son diamètre au sens de la distance de collaboration.
   Rem : La complexité de notre fonction distance est ici de $O(n^4)$, ce qui n'est vraiment pas optimal. Un parcours en largeur permet de calculer la distance entre deux sommets en $O(n+m)$ (pour les listes d"adjacences) ou $O(n^2)$ (pour une matrice d'adjacence) avec n le nombre de sommets et m le nombre d'arêtes. Notre méthode ici n'est donc pas optimale, mais en contrepartie, elle donne plus d'information que la seule distance. Nous verrons les parcours en largeur de graphe au prochain cours !