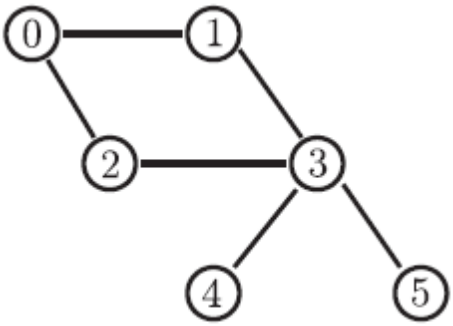
Exercice 1 : **Prise en main**



Partie 1 : Matrice d'adjacence

La matrice d'adjacence d'un graphe $G = (S; A)$ à n sommets est une matrice (ou un tableau de tableaux...)
à n lignes et n colonnes.
La case $(i; j)$ contient un 1 si les sommets i et j sont reliés par une arête, et un 0 sinon. On considère qu'un sommet n'est pas relié à lui-même, la diagonale $(i; i)$ ne contient donc que des 0.

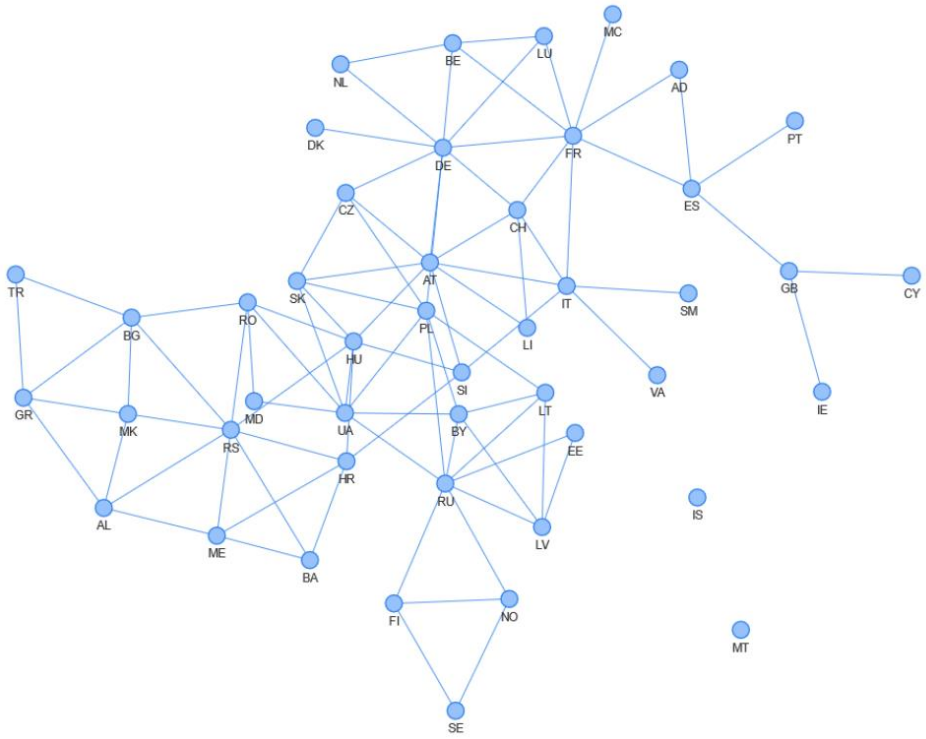
Q 1 : Représentez le graphe ci-contre sous la forme d'une matrice d'adjacence notée `M`.
Q 2 : Écrire une fonction `voisins(M, i)` qui prend en argument une matrice `M` et un sommet `i` et renvoie la liste des voisins de `i`.
Q 3 : Écrire une fonction `degre(M, i)` qui prend en argument une matrice `M` et un sommet `i` et renvoie le degré du sommet `i`.
Q 4 : Écrire une fonction `nb_aretes(M)` qui renvoie le nombre d'arêtes du graphe.

Partie 2 : Listes d'adjacence

Une autre manière de représenter un graphe est de lister les voisins de chacun de ses sommets. Ainsi, la liste d'adjacence d'un graphe à n sommets est une liste de n listes. Chaque sous-liste correspond à un sommet i et contient les voisins de i.
Q 5 : Donnez la liste d'adjacence du graphe de la partie 1, noté `L`.
Q 6 : Écrivez deux fonctions `liste_en_matrice(L)` et `matrice_en_liste(M)` qui permettent de convertir un graphe sous forme de liste en matrice et réciproquement. On pourra utiliser la fonction `voisins(M, i)`.

Exercice 2 : **Pays frontaliers en Europe**

Dans ce graphe, les pays en Europe sont les sommets et les arêtes correspondent à une frontière terrestre[1] en Europe.



[1] On n'a donc pas considéré la frontière entre la France et les Pays-Bas à Saint-Martin, mais on a bien considéré la frontière entre la Grande-Bretagne et l'Espagne à Gilbratar et la frontière entre Chypre et les bases britanniques d'Akrotiri et Dhekelia

Un fichier `Europe_etudiants.py` est disponible sur le site. Il définit la liste des pays par une liste de 46 chaines de caractères (la chaine de caractère est de longueur 2[2]) et une liste de listes correspondant aux 87 frontières (l'ordre est alphabétique, la frontière franco-belge est ainsi codée par la liste `['BE', 'FR']`).

Il crée également un graphe non orienté en utilisant la bibliothèque `networkx` et le dessine en utilisant `matplotlib`.

Partie 1 : Manipulation du graphe

Q 1 : Créer une liste d'adjacence L à partir des informations fournies.

Q 2 : Vérifier que `L[pays.index("FR")]` renvoie la liste `[1, 2, 5, 11, 19, 23, 27, 42]` (à l'ordre près). Comment obtenir le nom des pays voisins à la France à partir de cette liste ?

Q 3 : Écrire une fonction `verifL_GNO(L)` qui vérifie bien que `L` correspond bien à un graphe non orienté (renvoie `True` si c'est le cas, `False` sinon). Cela doit donc vérifier que, si `s2` appartient bien à la liste `L[s1]`, alors `s1` appartient bien à la liste `L[s2]`. Tester votre liste d'adjacence.

Q 4 : Créer un dictionnaire d'adjacence `d` à partir des informations fournies.

Q 5 : Vérifier que `d["FR"]` renvoie bien `['DE', 'AD', 'BE', 'ES', 'IT', 'LU', 'MC', 'CH']`, ce qui se traduit par :
les voisins de la France sont : l'Allemagne, Andorre, l'Espagne, l'Italie, le Luxembourg, Monaco et la Suisse.

Q 6 : Écrire une fonction `verifD_GNO(d)` qui vérifie bien que `d` correspond bien à un graphe non orienté. Tester votre dictionnaire d'adjacence.

Q 7 : Créer une `M` matrice d'adjacence associée au graphe. On utilisera la convention suivante : l'index de ligne correspond au pays de départ de l'arrête (ce pays a le même index dans la liste `pays`). L'index de colonne correspond au pays d'arrivée de l'arrête.

Q 8 : Vérifier que `M[14][5]` retourne bien 1 et que `M[35][14]` retourne bien 0.

Q 9 : Écrire une fonction `verifM_GNO(M)` qui vérifie bien que `M` correspond bien à un graphe non orienté. Tester votre matrice d'adjacence.

Partie 2 : Parcours du graphe

On souhaite parcourir le graphe au départ du sommet « FR ».

Q 10 : Ecrire un programme qui utilise un parcours en largeur pour afficher les pays accessibles depuis la France (directement et indirectement).

Q 11 : Compléter ce programme pour qu'il affiche ensuite les pays qui ne sont pas accessibles depuis la France.

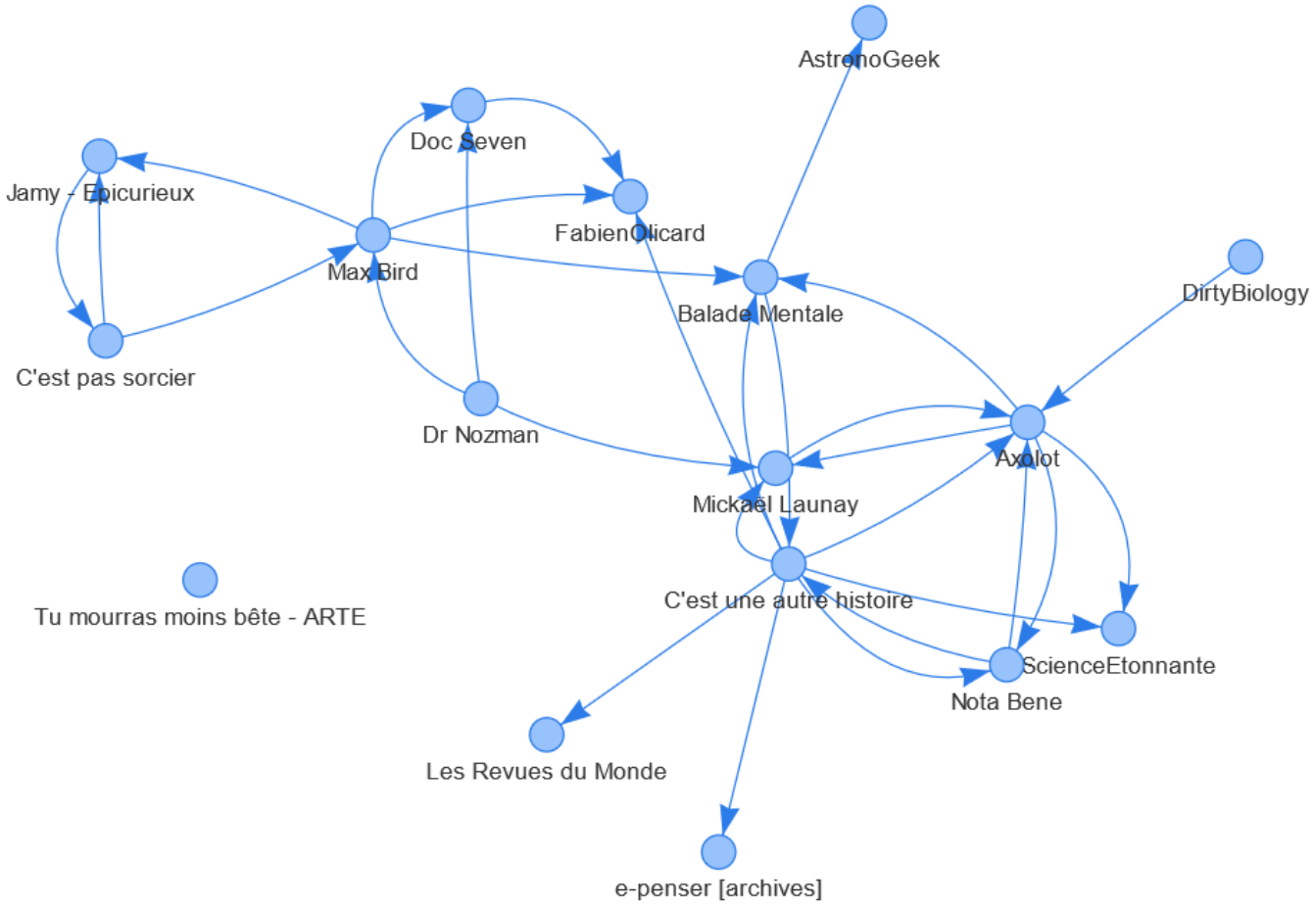
Q 12 : Ecrire un nouveau programme qui réalise cette fois un parcours en profondeur.

(cet exercice est fortement inspiré de celui proposé par T. Kovaltchouk)

[2] car on utilise la norme ISO 3166-1 alpha-2

Exercice 3 : **Vulgarisateurs francophones sur YouTube**

Dans ce graphe, des vulgarisateurs francophones à large audience (supérieur à 500 000 abonnés) sont les sommets et les arcs correspondent à un abonnement visible ou une recommandation sur leur page.



Un fichier `vulga_etudiants.py` est disponible sur le site. Il définit la liste des sommets (une liste de chaine de caractères) et une liste d'arcs (liste de listes de chaine de caractères). Il crée également un graphe orienté en utilisant la bibliothèque `networkx` et le dessine en utilisant `matplotlib`.

Q 1 : Écrire une fonction `dictAdj(s, a)` qui prend en entrée une liste de sommets `s` et une liste d'arcs `a` et renvoie le dictionnaire d'adjacence correspondant.

Utiliser la commande :
`G1 = nx.from_dict_of_lists(dictAdj(sommets, arcs), create_using=nx.DiGraph)`
pour créer un nouveau graphe en utilisant votre fonction. Dessiner ce graphe pour vérifier votre fonction.

Q 2 : Quelle est la chaine qui fait le plus de publicité ? Combien recommande-t-elle de collègues ?

Q 3 : Écrire une fonction `listeDeListes(s, a)` qui prend en entrée une liste de sommets `s` et une liste d'arcs `a` et renvoie la matrice d'adjacence correspondante sous la forme d'une liste de listes. Il sera utile d'utiliser un dictionnaire de numérotation `num` tel que `num[si] = i` (renvoie l'indice du sommet).

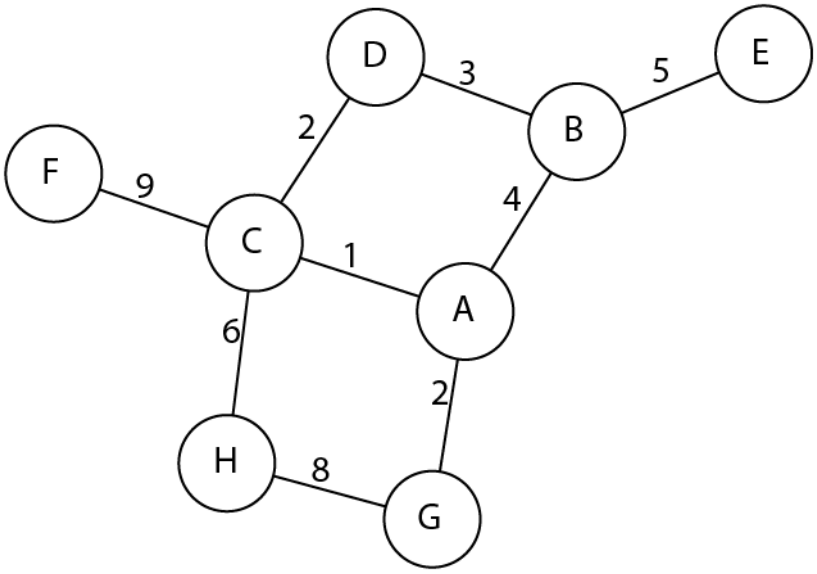
Utiliser la commande :
`G2=nx.from_numpy_matrix(np.array(listeDeListes(sommets,arcs)),create_using=nx.DiGraph)`
pour créer un nouveau graphe en utilisant votre fonction. Dessiner ce graphe pour vérifier votre fonction.

Q 4 : Quelle est la chaine qui reçoit le plus de publicité ? Combien de collègues la recommandent ?

Q 5 : En utilisant un parcours de graphe, tester s'il existe un chemin entre « Jamy » et « Axolot ». Tester s'il existe un chemin inverse.

Exercice 4 : **Plus court chemin : mise en œuvre sur un cas simple**

On considère le graphe suivant :



Son implémentation en machine est donnée sous la forme d'un dictionnaire d'adjacence dans le fichier réponse fourni sur le site http:\\touron.tech.

Q 1 : A la main, déterminer le chemin le plus court entre le sommet H et le sommet E.
Q 2 : Créer la liste contenant le nom des nœuds.
Q 3 : Ecrire une fonction qui, à partir d'un dictionnaire d'adjacence, renvoie la matrice d'adjacence associée.
Q 4 : Ecrire une fonction qui, à partir d'une matrice d'adjacence, un numéro de sommet de départ et un numéro de sommet d'arrivée, renvoie le chemin entre ces sommets, et la longueur totale du chemin. Cette fonction s'appuiera sur l'algorithme de Dijkstra.
Q 5 : Vérifier le résultat en comparant avec la question 1.

Exercice 5 : **Cas plus compliqué**

Le graphe représenté page suivante, et implémenté dans le fichier réponse, correspond à un graphe d'un labyrinthe d'un jeu vidéo. L'objectif est de rejoindre le plus vite possible les deux extrémités : « Sortie » et « Mother_Brain ».

Q 1 : En utilisant les fonctions définies précédemment, afficher le chemin le plus court entre ces deux sommets.