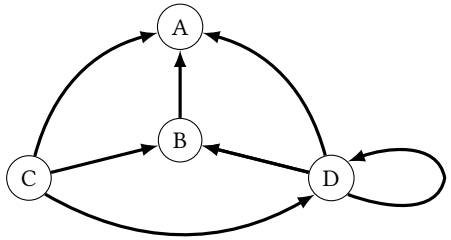
Exercice 1. Pour le graphe orienté : $\mathscr{G} = (\mathscr{S}, \mathscr{A})$ avec :

$\mathscr{S} = \{A, B, C, D\}$ $\mathscr{A} = \{(C,A); (C,B); (C,D); (B,A); (D,A); (D,B), (D,D)\}$



	A	B	C	D
A	0	0	0	0
B	1	0	0	0
C	1	1	0	1
D	1	1	0	1

On définira la liste de ses sommets en déclarant :
`Sommets = ['A', 'B', 'C', 'D']`

1. Déclarer le graphe à l'aide d'une liste d'adjacence `L`.
2. Déclarer le graphe à l'aide d'une matrice d'adjacence `M`.

Dans chacune des deux questions suivantes, on apportera deux solutions, l'une pour une liste d'adjacence, l'autre pour une matrice d'adjacence ; toutes les fonctions devront fonctionner pour un graphe quelconque.

3. Ecrire une fonction prenant en argument la liste `L` (respectivement la matrice `M`) et la liste `Sommets` et qui renvoie la liste des arêtes du graphes :
   `[('B','A'), ('C','A',), ...]`
4. Ecrire une fonction prenant en argument la liste `L` (respectivement la matrice `M`) et qui renvoyé `True` ou `False` selon si le graphe est non orienté.
5. Écrire deux fonctions `L2M` et `M2L` prenant en argument la liste `L` (respectivement la matrice `M`) et renvoyant la matrice d'adjacence (respectivement la liste d'adjacence) du même graphe.
6. Ecrire une fonction prenant en argument la liste d'adjacence, la liste des sommets, et deux sommets, et qui renvoie la distance entre ces deux sommets dans le graphe. Si aucun chemin ne relie les deux sommets la fonction ne renverra rien.

Exercice 2. Composantes connexes par un parcours en largeur.

On souhaite écrire une fonction qui détermine la composante connexe d'un sommet dans un graphe non orienté. Un graphe non orienté étant donné par sa matrice d'adjacence (tableau numpy), le sommet s par son indice, la fonction renverra la liste des sommets (leur indice) qui sont dans la même composante connexe que s.

Pour cela on applique un parcours en largeur du graphe. On constitue une liste `L` en commençant à y insérer le sommet s, puis tous les sommets adjacents à s, puis tous les sommets adjacents à un sommet de `L`, etc., ceci tant que la liste augmente.

Pour l'efficacité de l'algorithme, il faut prendre garde à la façon dont la liste `L` est conçue et augmentée sans créer de doublon.

1. Écrire une fonction prenant en paramètre la matrice d'adjacence d'un graphe orienté, et un sommet s (par un entier), et qui renvoie la liste des sommets adjacents à s.

La fonction `fusionner(L1,L2)` prend en paramètres deux listes d'entiers triés dans le sens croissant, et qui renvoie la liste d'entiers fusionnés, ordonnés dans le sens croissant et sans doublon.

```
def fusionner(L1,L2):
    L = []
    n1, n2 = len(L1), len(L2)
    i1 = i2 = 0
    while i1<n1 and i2<n2:
        if L1[i1] < L2[i2]:
            L.append(L1[i1])
            i1 += 1
        elif L1[i1] == L2[i2]:
            L.append(L1[i1])
            i1 += 1
            i2 += 1
        else:
            L.append(L2[i2])
            i2 += 1
    L += L1[i1:]
    L += L2[i2:]
    return L
```

2. Écrire la fonction principale `composanteConnexe(A,i)` qui prend en paramètres la matrice d'adjacence `A` et un sommet (entier) `i`, et qui renvoie la liste des sommets de la composante connexe du sommet. Elle utilisera la fonction `fusionner` pour insérer des sommets dans la liste `L`.
3. En déduire une fonction `connexe` prenant en paramètre la matrice d'adjacence d'un graphe non orienté et qui renvoie `True` selon si le graphe est ou non connexe.