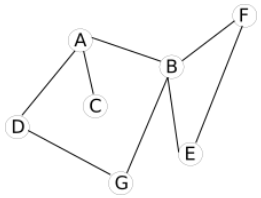
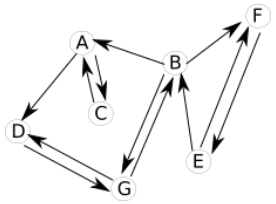
SDD_Graphes_Programmation_Exercices

Exercice 1 : Parcours en largeur / profondeur

1/ **Donner** les parcours en **profondeur** / **largeur** à partir du sommet A du graphe à droite.

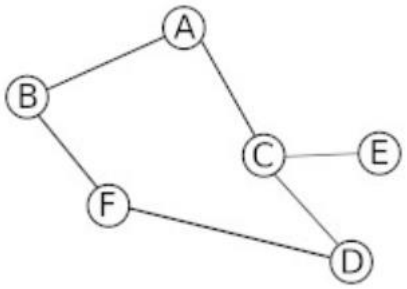


2/ **Même question** pour le graphe suivant à droite.



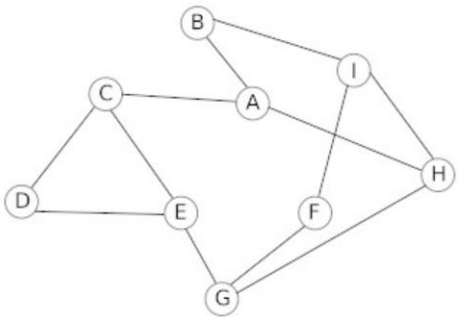
Exercice 2 : détection de cycle

En partant du sommet A, déterminer la **présence d'un cycle** ou pas **en appliquant l'algorithme**.



Exercice 3 : implémentation en Python

Proposer une implémentation en langage Python du graphe à droite (jeu de test seulement).
Aide : on pourra partir d'une des deux classes Graphe du cours



Exercice 4 :

Soit la matrice d'adjacence suivante G

1/ Faire un schéma du **graphe** associé (noté G).
2/ **Implémenter** ce graphe en langage Python (jeu de test seulement).

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

Voici le programme suivant :

```
g1 = {'A':['B','C'], 'B':['A'], 'C':['A','D'], 'D':['C']}

def myst(G,s):
    noir = []
    pile = [s]
    while len(pile) > 0 :
        u = pile.pop()
        if u not in noir :
            noir.append(u)
            for v in G[u]:
                pile.append(v)
    return noir


L = myst(g1,'A')
```

3/ Que **vaut** la variable *L* après exécution du programme ?

4/ On considère un graphe G muni des méthodes du cours (*voisins(self,s)*, *sommets(self)* etc.)
Compléter alors le programme suivant indiquant la présence d'un cycle ou non.

```
def cycle(G):
    s = random.choice(list(G.keys()))
    p = []
    p.append(s)
    noir=[]
    while len(p)>0:
        u = p.pop()
        for v in ……:
            if v not in noir:
                p.append(....)
        if u in …….:
            return True
        else :
            noir.append(u)
    return ……
```