C) Parcours en profondeur (DFS) :

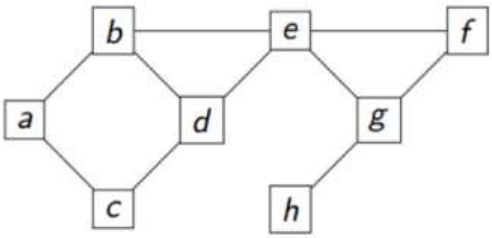
Principe : Pour parcourir un graphe en profondeur :

- On explore les sommets du graphe **en passant de sommet en sommet en suivant l'un des voisins** (ou successeurs) et en marquant les sommets visités.
- Lorsqu'il n'y a plus de sommets accessibles (i.e. non encore visités), on revient au sommet précédent.

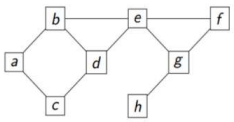
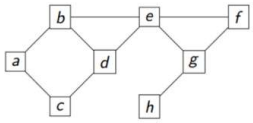
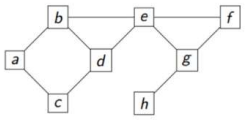
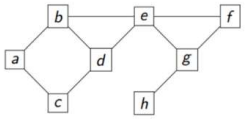
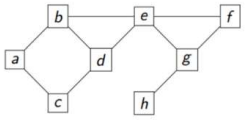
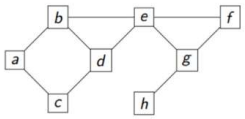
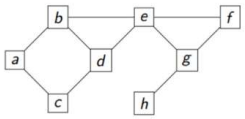
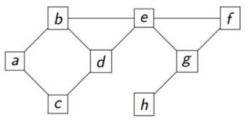
Dans l'algorithme, on utilise une **pile**. Attention : Suivant la façon dont on **empile** les voisins (ou successeurs), on peut trouver des ordres différents dans la liste des sommets lors du parcours en profondeur d'abord.

Le parcours en profondeur est le parcours typique que l'on fait lorsqu'on explore un labyrinthe.

Prenons en exemple ce graphe :



Complèter chaque étape ci-dessous

	s = b visités =[b] p = [a,d,e <
	s =e visités =[b,e] p = [a, d, f , g <
	s = g visités =[b,e,'] p = [a, d, ...
	
	
	
	
	

Algorithme

Fonction parcours_profondeur(graphe,s_depart)
  créer une pile p
  empiler …… dans p
  visités ← []
  Tant que …… :
    Dépiler …… dans une variable s
    Si …… n'est pas dans ……
      ……
    Fin si
    Pour chaque voisin v de …….
      Si …… n'est pas dans …….. ni dans ……
        ……
      Fin si
    Fin pour
  Fin Tant que
  Retourner ……

On en déduit le parcours en profondeur en partant du sommet 'b' : ……

Donner un parcours en profondeur en partant de 'd' :

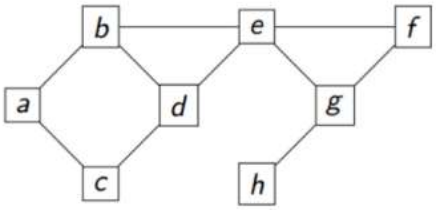
……

Dans le notebook, écrire le code d'une fonction Python nommée parcours_en_profondeur (graphe,s_depart) qui parcourt en profondeur un graphe à partir du sommet s_depart et dont sa liste d'adjacence graphe est représentée par un dictionnaire.

On utilisera au mieux une liste pour représenter une pile

D) Bonus : Recherche de chemins dans un graphe entre deux sommets :

Exemple : dans le graphe ci-contre,
lister les chemins possibles entre le sommet 'c' et le sommet 'g'



Pour quel type de graphe est-on sûr qu'il existe un chemin entre deux sommets quelconque d'un graphe ? ……

On connait deux types de parcours d'un graphe (largeur et profondeur), lequel privilégier pour la recherche de chemins ?

……

On va donc utiliser le même algorithme avec une …….. où l'on e…..ilera un couple (sommet, chemin) où chemin est un tableau des sommets visités à partir du sommet départ.

Dans un chemin ['a','b','c',…..] on dira que 'c' est le successeur du sommet 'b'

- **Fonction unchemin (g : un graphe, départ : sommet, arrivée : sommet)**

```
Début
    Créer une pile p avec le tuple ( ……., ……..)
    Tant que la pile est non vide
        sommet, chemin ← dépiler la pile
        Créer un tableau voisins des successeurs de sommet qui ne sont pas dans chemin
        Pour chaque successeur de voisins
            si successeur = …… alors
                retourner …….
            Sinon
                empiler ( ……., ……..)
            fin si
        Fin pour
    Fin tant que
    Retourner …… # si pas de chemin
Fin
```

Implémenter cette fonction en python et faire des jeux de tests

- Ecrire une fonction **touschemin(g,depart,arrivée)** qui donne tous les chemins entre les sommets départ et arrivée
  Que modifier dans l'algorithme précédent ?

- Ecrire la fonction **pluscourtchemin (g,depart,arrivée)** qui donne le plus court des chemins entre les sommets départ et arrivée