Parcours d'un graphe en profondeur

Objectif : Comprendre le principe du parcours en profondeur et en largeur dans un graphe.

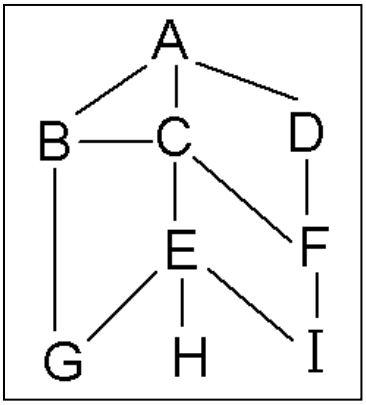
Parcours en profondeur

Vidéo pour bien comprendre : (6 premières minutes suffisent, 15 mn total pour bien assimiler)
https://www.youtube.com/watch?v=kcedjJOjDpg

Utilisation de la pile pour le backtrack :
https://www.youtube.com/watch?v=-wPLAsAuenA

Programmation :

A partir du graphe non orienté suivant, compléter la matrice d'adjacence :



	A	B	C	D	E	F	G	H	I
A		1	1	1					
B									
C									
D									
E									
F									
G									
H									
I									

On mettra un 1 pour indiquer une liaison entre deux sommets, 0 sinon. Se référer au cours précédent sur les graphes si nécessaire.

Donner la liste des sommets visités lors de deux parcours en profondeur possibles en partant du sommet A.

Parcours 1 possible :

Parcours 2 possible :

Taper le programme (voir page 2) de parcours en profondeur et tester le en enregistrant avec votre matrice d'adjacence. Vérifier que vous obtenez bien le résultat ci-dessous :

Donner la liste des sommets visités ainsi que l'arbre correspondant lors d'un parcours en profondeur possible en partant du sommet I

A B C E G H I F D

Donner l'arbre obtenu correspondant

```python
# Parcours graphe

class noeud:

    def __init__ (self,numero,liste_sommets):

        self.etiquette = numero                # numero du sommet
        self.couleur = "blanc"                 # couleur arbitraire
        self.nb_sommets = len(liste_sommets)   # nombre de sommets du graphe
        self.liste_pointeur_voisins =[]


        for i in range(self.nb_sommets):
            self.liste_pointeur_voisins = self.liste_pointeur_voisins + [None]

        print (self.liste_pointeur_voisins)

# matrice d'adajacence

#     a b c d e f g h i
M = [[0,1,1,1,0,0,0,0,0], #a
     [1                0], #b
     [1                0], #c
     [1                0], #d
     [0                1], #e
     [0                1], #f
     [0                0], #g
     [0                0], #h
     [0,0,0,0,1,1,0,0,0]] #i


# création du graphe avec la liste des sommets
sommets = ["A","B","C","D","E","F","G","H","I"]
graphe = [noeud("A",sommets),noeud("B",sommets),noeud("C",sommets),noeud("D",sommets),
          noeud("E",sommets),noeud("F",sommets),noeud("G",sommets),noeud("H",sommets),noeud("I",sommets)]

# création des liaisons entre les sommets
for ligne in range(len(M)):
    for colonne in range(len(M)):
        if M[ligne][colonne] == 1:
            graphe[ligne].liste_pointeur_voisins[colonne]=graphe[colonne]


 # Algo parcours en profondeur (version récursive)
    #u : noeud
    #v : noeud

def parcours_profondeur(u):
    u.couleur = "noir"
    print (u.etiquette, end=" ")
    for v in                    :
        if v != None and v.couleur !="noir" :
            parcours_profondeur(v)

u = graphe[0]           # noeud A
parcours_profondeur(u)
```

Matrice d'adjacence à compléter

Compléter pour parcourir tous les voisins du nœud u en accédant à l'attribut liste_pointeur_voisins de u

Si vous avez tout terminé, entraînez vous en partant d'un nœud quelconque pour visiter le graphe ne profondeur. Comparez avec ce que propose le programme pour le même noeud.