

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)

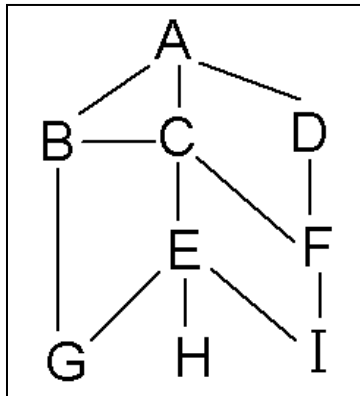
<https://www.youtube.com/watch?v=kcedjJOjDpg>

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

```

1  # Parcours graphe
2
3  class noeud:
4
5      def __init__(self, numero, liste_sommets):
6
7          self.etiquette = numero          # numero du sommet
8          self.couleur = "blanc"          # couleur arbitraire
9          self.nb_sommets = len(liste_sommets)  # nombre de sommets du graphe
10         self.liste_pointeur_voisins = []
11
12
13         for i in range(self.nb_sommets):
14             self.liste_pointeur_voisins = self.liste_pointeur_voisins + [None]
15
16         print (self.liste_pointeur_voisins)
17
18     # matrice d'adjacence
19
20     #      a b c d e f g h i
21     M = [[0,1,1,1,0,0,0,0,0], #a
22          [1,0,0,0,0,0,0,0,0], #b
23          [1,0,0,0,0,0,0,0,0], #c
24          [1,0,0,0,0,0,0,0,0], #d
25          [0,0,0,1,0,0,0,0,0], #e
26          [0,0,0,1,0,0,0,0,0], #f
27          [0,0,0,0,1,0,0,0,0], #g
28          [0,0,0,0,1,0,0,0,0], #h
29          [0,0,0,0,1,1,0,0,0]] #i
30
31
32     # création du graphe avec la liste des sommets
33     sommets = ["A","B","C","D","E","F","G","H","I"]
34     graphe = [noeud("A",sommets),noeud("B",sommets),noeud("C",sommets),noeud("D",sommets),
35              noeud("E",sommets),noeud("F",sommets),noeud("G",sommets),noeud("H",sommets),noeud("I",sommets)]
36
37     # création des liaisons entre les sommets
38     for ligne in range(len(M)):
39         for colonne in range(len(M)):
40             if M[ligne][colonne] == 1:
41                 graphe[ligne].liste_pointeur_voisins[colonne]=graphe[colonne]
42
43
44     # Algo parcours en profondeur (version récursive)
45     #u : noeud
46     #v : noeud
47
48     def parcours_profondeur(u):
49         u.couleur = "noir"
50         print (u.etiquette, end=" ")
51         for v in :
52             if v != None and v.couleur != "noir" :
53                 parcours_profondeur(v)
54
55     u = graphe[0]          # noeud A
56     parcours_profondeur(u)
57

```

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 en profondeur. Comparez avec ce que propose le programme pour le même nœud.