

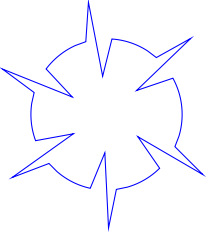
CHAPITRE
7 PARCOURS DE GRAPHES 

Table des matières

1	Parcours de graphes.....	2
1.1	Parcours en profondeur d'un graphe G	2
1.2	Parcours en largeur d'un graphe G	4
2	Implémentation en Python des algorithmes.....	5
3	Le coin des exercices!	6

1 Parcours de graphes

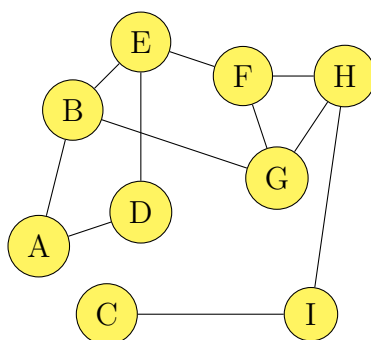
Un graphe G étant donné, on appelle **parcourir** un graphe la tâche qui consiste à explorer tous les sommets du graphe. Le parcours infixe d'un arbre binaire partage la même volonté : cependant dans un graphe, il n'y a pas de sommet « racine » qui marquerait le début du parcours. Le parcours débutera donc par un sommet quelconque.

On doit s'assurer de ne pas tourner en rond lorsqu'on explore les sommets d'un graphe : il faudra donc garder en mémoire les sommets visités.

Il existe deux algorithmes de parcours de graphe que nous allons exposer ici...

1.1 Parcours en profondeur d'un graphe G

Soit G le graphe suivant :

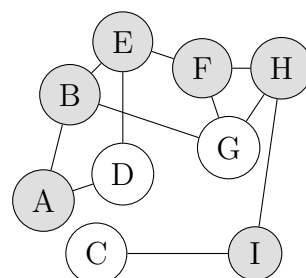
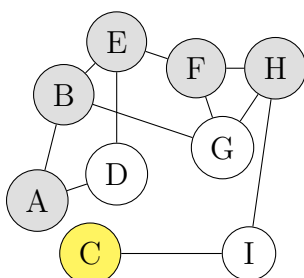
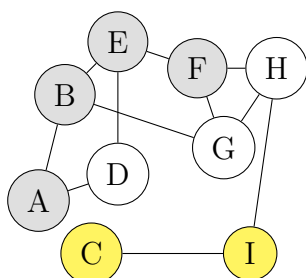
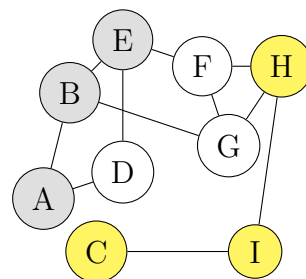
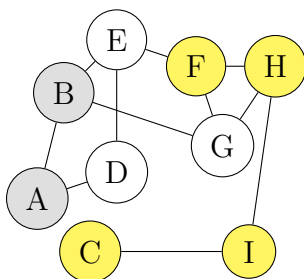
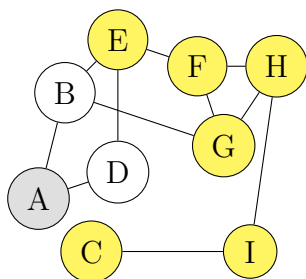


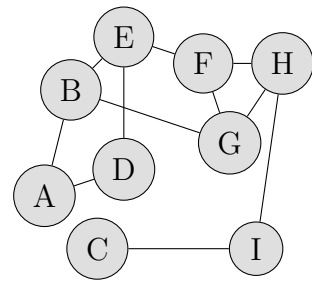
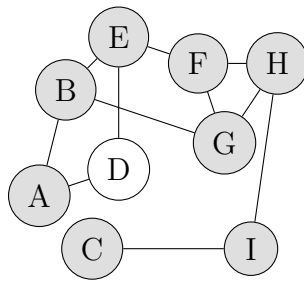
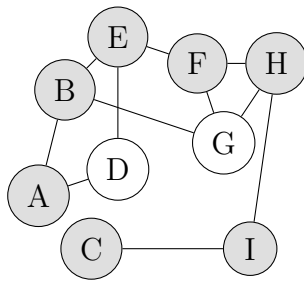
Dans le parcours en profondeur DFS (Depth First Search), on explore les voisins du sommet de départ, puis les voisins du premier voisin, puis les voisins du premier voisin, du premier voisin jusqu'à l'exploration de tous les sommets.

Les illustrations ci-dessous montrent les diverses étapes du DFS :

⇒ les sommets visités sont en gris

⇒ les voisins en blanc





Le parcours en profondeur partant de A , est donc : $A - B - E - F - H - I - C - G - D$

MAIS, ce choix de profondeur ne dépend finalement que du choix du premier voisin : j'ai pris le parti de désigner celui-ci comme étant celui le plus ... haut, choix totalement arbitraire!

Le parcours aurait donc pu être : $A - D - E - F - G - H - I - C - B$.

En pratique nous allons utiliser une pile P pour conserver les voisins de chaque sommets visités : lorsque la pile est vide, la conclusion est rapide : tous les sommets ont été visités !

Exercice n°1

Donner un parcours en profondeur du graphe précédent en partant du point H en montrant l'état de la pile P .

Ces manipulations simples permettent de construire un algorithme dont les étapes pourraient être :

Entrée(s) un graphe G et un sommet s de G

Sortie(s) une liste de sommet

empiler(s, P)

tant que P est non vide **faire**

$u =$ depiler(P)

si u n'est pas marqué **alors**

 marquer u

 afficher u

pour chaque voisin v de u **faire**

 empiler(v, P)

fin du pour

fin du si

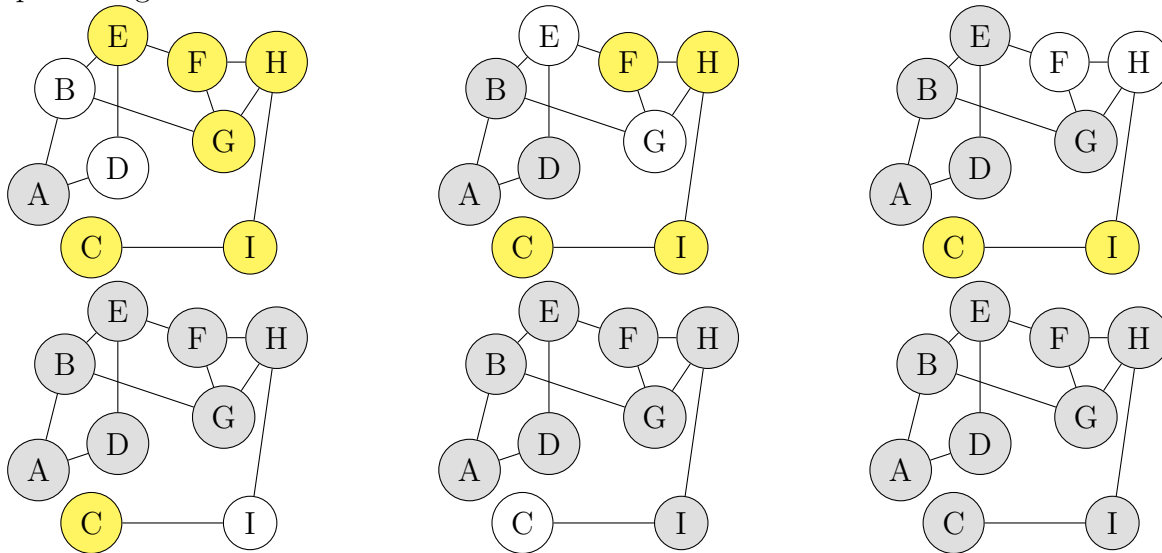
fin du tant que

Algorithme 1 : PARCOURS EN PROFONDEUR D'UN GRAPHE G

Pour marquer un sommet on pourra par exemple associer un booléen à chaque sommet dont la valeur donne l'état du sommet : marqué(True) ou non(False)! Notons enfin que le parcours en profondeur est récursif! On pourra donc utiliser une fonction récursive pour l'implémenter.

1.2 Parcours en largeur d'un graphe G

Dans le parcours en largeur d'un graphe(Breadth First Search), on explore tous les voisins du sommets de départ, puis tous les voisins de chaque voisin, etc... Une illustration vaut mieux qu'un long discours :



Le parcours en largeur partant de A est donc : $A - B - D - E - G - F - H - I - C$

Puisqu'on traite en priorité les premiers voisins visités (et non pas les derniers dans le DFS...), l'algorithme de parcours en largeur repose sur l'utilisation d'une **file** pour garder en mémoire les voisins non visités et à traiter.

Entrée(s) un graphe G et un sommet s de G

Sortie(s) une liste de sommet

enfiler(s, F)

tant que F est non vide **faire**

$u = \text{defiler}(F)$

 afficher u

pour chaque voisin v de u **faire**

si v n'est pas marqué **alors**

 marquer v

 enfiler(v, F)

fin du si

fin du pour

fin du tant que

Algorithme 2 : PARCOURS EN LARGEUR D'UN GRAPHE G

2 Implémentation en Python des algorithmes

On suppose connu le graphe G suivant :

```
1 G = [  
2 [1, 6, 7], # voisins de 0  
3 [0, 2, 4, 5, 6], # voisins de 1  
4 [1], # voisins de 2  
5 [5, 6, 8], # voisins de 3  
6 [1], # voisins de 4  
7 [1, 3, 6], # voisins de 5  
8 [0, 1, 3, 5, 7], # voisins de 6  
9 [0, 6, 8], # voisins de 7  
10 [3, 7] # voisins de 8  
11 ]
```

On rappelle que **parcourir un graphe**, consiste à visiter ses différents sommets et d'opérer une action tour à tour sur eux(comme les imprimer...). Le programme Python suivant permet le parcours d'un graphe G :

```

1 def visiter(k):
2     # action à effectuer dans le parcours lors de la visite du sommet k
3     print(k, end=' ')
4 def parcours(graphe):
5     n = len(graphe)
6     depuis = []
7     depuis.append(0)
8     dejavu = [False for k in range(n)]
9     while not(len(depuis) == 0):
10        k = depuis.pop()
11        if dejavu[k]:
12            continue
13        dejavu[k] = True
14        visiter(k)
15        for s in graphe[k]:
16            depuis.append(s)

```

L'appel de `parcours(G)` affiche la ligne 078365142. On a effectué un parcours en profondeur d'abord : on avance autant que possible, et on revient sur ses traces pour les sommets qu'on a laissés de côté préalablement.

3 Le coin des exercices !

Exercice n°2

Modifier le programme précédent pour obtenir un parcours en **largeur** d'abord.

Exercice n°3

Construire les fonctions `parcoursDFS(G,s)` et `parcoursBFS(G,s)` qui parcourent un graphe G à partir d'un sommet s en profondeur d'abord et en largeur ensuite. On utilisera les implémentations des Piles et Files rencontrées dans des chapitres précédents.

Ces fonctions retournent la liste des sommets parcourus.