Algorithmique	**Parcours d'un graphe**	Parcours en largeur, en profondeur

Parcourir un graphe consiste à lister tous ses sommets en une seule fois.

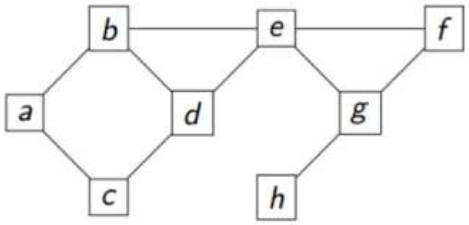
A) Parcours en largeur (BFS)

Principe : Parcourir en **largeur** un graphe à partir d'un sommet revient à visiter les voisins de ce sommet (c'est à dire les sommets à distance 1) puis les enfants des voisins (les sommets à distance 2 du point de départ) etc…

Idée : utiliser une file pour gérer les sommets.

1. On enfile le sommet de départ
2. On défile la file
3. On enfile les voisins du sommet défilé (à distance 1 ou sommets adjacents au sommet initial) s'ils n'ont pas été visités.
4. On réitère les points 2 et 3 tant que la file n'est pas vide.

Prenons en exemple ce graphe :



Fonction parcours_largeur (graphe,s_depart)

créer une file f

enfiler …… dans la f

visités ← […….]

Tant que ……:

défiler f dans s

Si …. n'est pas dans …….

Ajouter … à …….

Fin si

Pour chaque voisin v de ….

Si …… n'est ni dans …… ni dans ……

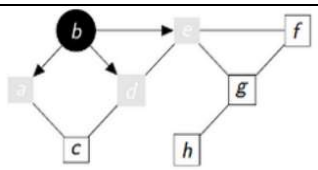
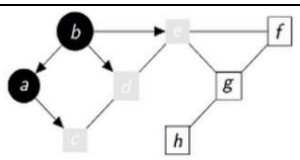
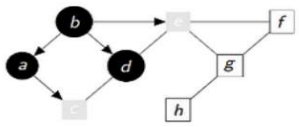
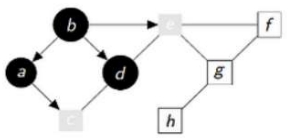
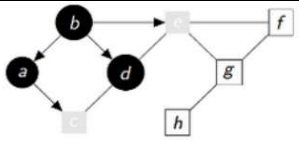
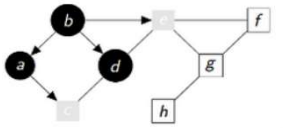
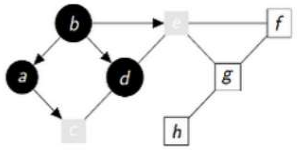
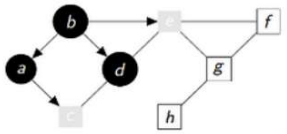
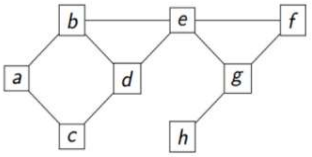
……

Fin si

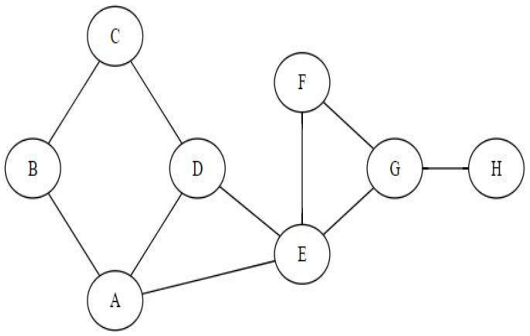
Fin pour

Fin tant que

Retourner visités

	Etape 1 : file= < b< visités =[b]		Etape 2 : s =b visités =[b] file= < a,d,e <
	Etape 3 : s = a visités =[b,a] f = < d, e, c <		Etape 4 : s = visités= file =
	Etape 4 : s= visités= file=		Etape 5 : S =c visités= File =
	Etape 6 : S= g visités= file=		Etape 7 : S= visités= file=
	Etape 8 : S= h visités= file=	Le parcours en largeur du graphe en commençant par le sommet 'b' est : ……	

Exercice : Soit le graphe G :



Que retourne parcours_largeur(G,'A') ?

…….

sommets								
Distance de A								

Que retourne parcours_largeur(G,'F') ?

……

sommets								
Distance de F								

Ouvrir le notebook parcours_graphes.ipynb , écrire le code d'une fonction nommée

parcours_en_largeur(graphe,s_depart) qui prend en paramètre d'entrée un graphe et un sommet s_depart et qui retourne dans une liste le parcours en largeur à partir du sommet s_depart .

On utilisera l'implémentation du graphe par sa liste d'adjacence graphe , représentée par un dictionnaire.

On utilisera une liste pour représenter une file avec f.pop(0) qui supprime et retourne le 1er élément

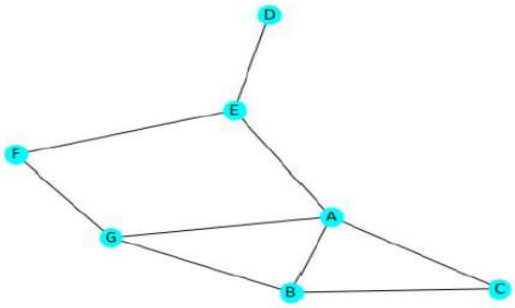
B) Applications du parcours en largeur à la recherche de cycle dans un graphe :

Une chaîne est une suite d'arêtes consécutives dans un graphe qui est désignée par la suite des sommets par lesquels elle passe. Un **cycle** est une chaîne qui commence et finit par le même sommet et qui ne passe pas plusieurs fois par la même arête. On parlera de circuit pour un graphe orienté.

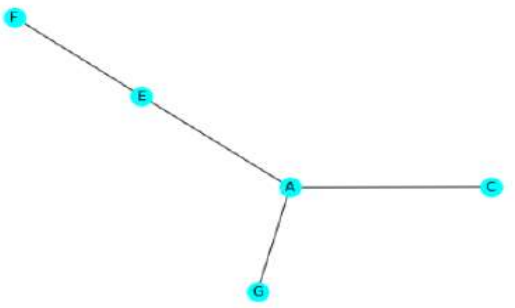
Algorithme de détection de cycle

On va utiliser un parcours en largeur d'abord que l'on va aménager pour répondre à notre problématique.
Pour déterminer si un graphe possède un cycle, il nous faut plus que le parcourir.
Ne devant pas repasser par une arête déjà visitée, il nous faut répertorier les parents de chaque sommet visité.

1. On place le sommet de départ dans la file
2. On met les voisins du sommet de départ dans la file s'ils n'ont pas été visités.
3. On défile le premier sommet entré : S'il a été déjà visité, on renvoie vrai, sinon on réitère l'opération jusqu'à ce que la file soit vide (on renvoie faux) ou que l'on renvoie vrai.



Etat de la file avant analyse	Nœuds visités
E	E
A D F	E A
D F C B G	E A D
F C G B G	E A D F
C G B G B	E A D F C
G B G B B	E A D F C G
B G B B: Algorithme stoppé : Nœud déjà dans visite	E A D F C G B True



Compléter :

Etat de la file avant analyse	Nœuds visités
E	E

Compléter le notebook en écrivant la fonction detection_cycle(g,s_depart)