Python - TP 8

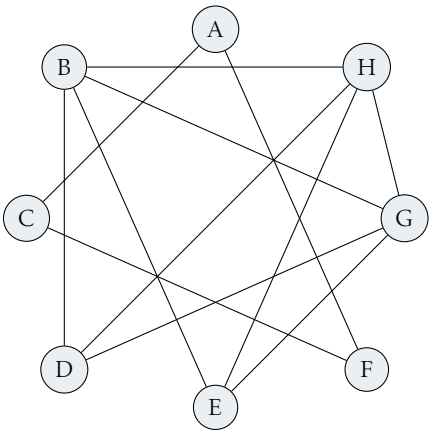
Graphes : algorithme de parcours en profondeur

Objectifs principaux :

1. implémenter un graphe par la liste des listes d'adjacence,
2. explorer le graphe en profondeur à partir d'un sommet pour en déterminer sa composante connexe.

Objectif secondaire
Obtenir toutes les composantes connexes d'un graphe.

On considère le graphe suivant :



Définition 1 - Liste d'adjacence

Soit $\mathcal{G} = (S, \mathcal{A})$ un graphe.
Soit $s \in S$. La liste d'adjacence de s est la liste de tous les sommets adjacents à s.

✘ Attention !
On veillera à toujours ordonner les listes d'adjacence et la liste des listes d'adjacences convenablement...

Petite remarque
On choisit de définir le graphe par *la liste des chaînes de caractères d'adjacence*, pour plus de légèreté à l'écriture et à la lecture...

1. **Création de la liste des listes d'adjacence.**
   - 1.a. Définir `S` la chaîne de caractère contenant les sommets du graphe.
   - 1.b. Créer une liste `Ladj` contenant autant de chaînes de caractères vides que de sommets du graphe.
   - 1.c. Remplir `Ladj` par la chaîne de caractère d'adjacence de chaque sommet du graphe.
     La liste `Ladj` contient alors toutes les listes d'adjacence du graphe (sous forme de chaînes de caractères).
2. **Exploration en profondeur d'un graphe à partir d'un sommet initialement choisi.** On rappelle le principe de cet algorithme d'exploration en profondeur :

> Soient alors $\mathcal{G} = (S, \mathcal{A})$ et s un sommet de $\mathcal{G}$.
> - si s n'a pas déjà été visité (ce qui, initialement, est le cas) on commence par visiter s,
> - puis, pour chaque sommet adjacent à s qui n'a pas encore été visité, noté t :
>   - ◇ on visite t,
>   - ◇ puis, pour chaque sommet adjacent à t qui n'a pas encore été visité, noté u :
>   - ⇝ on visite u,
>   - ⇝ puis ...

   - 2.a. Créer une liste `V`, initialement vide, contenant les sommets visités.
   - 2.b. Créer la fonction `explorer` telle que l'exécution de `explorer(Ladj,'A')` renvoie la liste des sommets reliés au sommet A (sa composante connexe).

Aide
On pourra utiliser la commande `S.index(s)` qui renvoie la position du caractère `s` dans la chaîne de caractère (ou liste) `S`.

```
S='ABCDEFGH'
Ladj=['CF','DEGH','AF','BHG','BHG','AC','BDEH','BDEG']

V=str()
def explorer(G,s):
    global V
    r=S.index(s)
    if s not in V:
        V=V+s
        for x in G[r]:
            if x not in V:
                explorer(G,x)
    return V
```

3. Pour finir : créer une fonction prenant en argument d'entrée la liste des listes d'adjacence d'un graphe et renvoyant ses composantes connexes.

```
S='ABCDEFGH'
Ladj=['CF','DEGH','AF','BHG','BHG','AC','BDEH','BDEG']

def explorer(G,s):
    global M,V
    r=S.index(s)
    if s not in V:
        V=V+s
        M=M+s
        for x in G[r]:
            explorer(G,x)
    return V

def compconnexe(G):
    global M,V
    M=str() # sommets marqués
    Lcomp=[] # liste composantes connexes
    for s in S:
        if s not in M:
            V=str() # visités depuis s
            Lcomp.append(explorer(G,s))
    return Lcomp
```