NUMERIQUE et SCIENCES INFORMATIQUES

Partie pratique

Classe Terminale de la voie générale

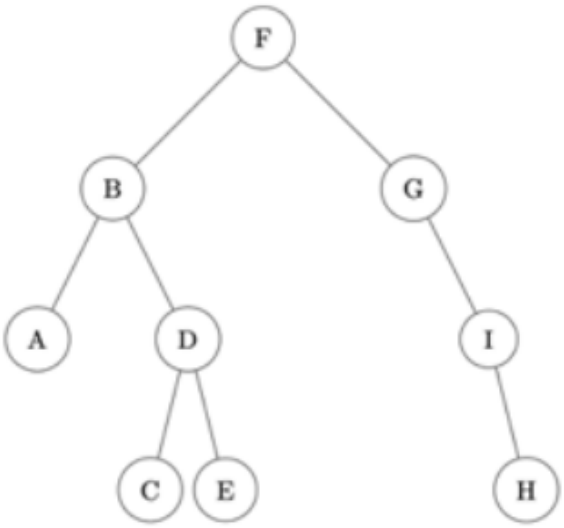
Sujet n°27

Exercice1 (correction) :

EXERCICE 1 (4 points)

Dans cet exercice, un arbre binaire de caractères est stocké sous la forme d'un dictionnaire où les clefs sont les caractères des nœuds de l'arbre et les valeurs, pour chaque clef, la liste des caractères des fils gauche et droit du nœud.

Par exemple, l'arbre



est stocké dans

```
a = {'F':['B','G'], 'B':['A','D'], 'A':['',''], 'D':['C','E'], \
     'C':['',''], 'E':['',''], 'G':['','I'], 'I':['','H'], \
     'H':['','']}
```

Écrire une fonction récursive `taille` prenant en paramètres un arbre binaire `arbre` sous la forme d'un dictionnaire et un caractère `lettre` qui est la valeur du sommet de l'arbre, et qui renvoie la taille de l'arbre à savoir le nombre total de nœud.

Réponses :

```
a = {'F':['B','G'], 'B':['A','D'], 'A':['',''], 'D':['C','E'], \
 'C':['',''], 'E':['',''], 'G':['','I'], 'I':['','H'], \
 'H':['','']}



def taille(arbre, lettre):
    fils_gauche = arbre[lettre][0]
    fils_droit = arbre[lettre][1]

    if fils_gauche != '' and fils_droit != '':
        return 1 + taille(arbre, fils_gauche) + taille(arbre, fils_droit)

    if fils_gauche != '' and fils_droit == '':
        return 1 + taille(arbre, fils_gauche)

    if fils_gauche == '' and fils_droit != '':
        return 1 + taille(arbre, fils_droit)

    else:
        return 1
```

a est un dictionnaire de liste :
Les deux premières lignes :

```
fils_gauche = arbre[lettre][0]
fils_droit = arbre[lettre][1]
```

Permettent de définir les deux fils, gauche et droit de la lettre, il va chercher la liste correspondante a la lettre dans le dictionnaire et attribue la première valeur au fils gauche et la seconde au fils droit.

```
if fils_gauche != '' and fils_droit != '':
    return 1 + taille(arbre, fils_gauche) + taille(arbre, fils_droit)
```

Ici le fils gauche et droit existe, donc la taille est égale à taille arbre + taille fils gauche + taille fils droit.

```
if fils_gauche != '' and fils_droit == '':
    return 1 + taille(arbre, fils_gauche)
```

Ici seul le fils gauche existe, donc la taille est égale à taille arbre + taille fils gauche.

```
if fils_gauche == '' and fils_droit != '':
    return 1 + taille(arbre, fils_droit)
```

Ici seul le fils droit existe, donc la taille est égale à taille arbre + taille fils droit.

```
else:
    return 1
```

Renvoie la taille de l'arbre si aucun fils n'existe.

Exercice 2 (correction) :

```
def tri_iteratif(tab):
    for k in range( ... , 0, -1):
        imax = ...
        for i in range(0 , ... ):
            if tab[i] > ... :
                imax = i
        if tab[imax] > ... :
            ... , tab[imax] = tab[imax] , ...
    return tab
```

Compléter le code qui doit donner :

```
>>> tri_iteratif([41, 55, 21, 18, 12, 6, 25])
[6, 12, 18, 21, 25, 41, 55]
```

Réponses :

```
def tri_iteratif(tab):
    for k in range( len(tab)-1 , 0, -1):
        imax = 0
        for i in range(0 , k ):
            if tab[i] > tab[imax]:
                imax = i
        if tab[imax] > tab[k] :
            tab[k] , tab[imax] = tab[imax] , tab[k]
    return tab
```

```
>>> tri_iteratif([41, 55, 21, 18, 12, 6, 25])
[6, 12, 18, 21, 25, 41, 55]
```

```
for k in range( len(tab)-1 , 0, -1):
```

On ajoute le nombre de valeur de tab et lui soustrait 1.

```
imax = 0
```

On donne la valeur 0 a i max.

```
for i in range(0 , k ):
    if tab[i] > tab[imax]:
        imax = i
```

Si i est plus grand que i max, i max = i.

```
if tab[imax] > tab[k] :
    tab[k] , tab[imax] = tab[imax] , tab[k]
```

Si le nombre i max dans tab est plus grand que le nombre k dans tab les deux valeurs sont inversées i max devient k et inversement.

```
return tab
```

Renvoie tab.