

## A) Arbres binaires : occurrences ... ( 2 points)

Soit l'arbre  $B$  défini, sous forme d'occurrences et représenté à l'aide d'une chaîne de caractères de la manière suivante :

$B = "0,1,00,01,10,11,001,101,1010,1011"$

Sans représenter l'arbre  $B$ , pour déterminer :

- 1)  $T(B)$ , il suffit de compter les virgules présentes dans la chaîne et d'y ajouter  $1$ ,
- 2)  $H(B)$ , il suffit de compter le nombre de  $0$  et de  $1$  de la dernière occurrence de la chaîne ( $1011$  dans l'exemple),
- 3) qu'un nœud est une feuille, il faut que son occurrence n'en préfixe pas une autre. Il suffit donc de parcourir le reste de la chaîne en vérifiant que l'occurrence de ce nœud ne se trouve pas au début des occurrences suivantes,
- 4)  $LC(B)$ , il suffit de compter le nombre total de  $0$  et de  $1$  se trouvant dans la chaîne,

## B) Matrices : transposée ... (3 points)

### Spécifications :

La fonction **transpose(M)** construit et retourne la matrice transposée de la matrice non vide **M**.

La fonction **test** suivante :

```
def buildTranspose (M):  
    (l, c) = (len(M), len (M [0]) )  
    R = []  
    for i in range (c):  
        L = []  
        for j in range (l):  
            L.append (M[j][i])  
        R.append (L)  
    return R
```

### C) Matrices : symétrie ... (5 points)

#### Spécifications :

La fonction **symetric(M)** vérifie si la matrice **M** est symétrique selon un axe horizontal (symétrie verticale).

```
# with a bool variable
def symetric (M):
    (l, c) = (len(M), len(M [0]))
    ldiv2 = l // 2
    i = 0
    stop = True
    while i < ldiv2 and stop :
        j = 0
        while j < c and test:
            test = M[i][j] == M[l-i-1][j]
            j += 1
        i += 1
    return not stop
```

```
# without a bool variable
def symetric (M):
    (l, c) = (len(M), len(M [0]))
    ldiv2 = l // 2
    (i, j) = (0, c)
    while i < ldiv2 and j == c :
        j = 0
        while j < c and M[i][j] == M[l-i-1][j]:
            j += 1
        i += 1
    return j == c
```

## A) Procédé de Kaprekar... (16 points)

### 1) Test ... (1 point)

La fonction **test** suivante :

```
def test(x, L):  
    i = len(L) - 1  
    while i >= 0 and L[i] != x:  
        i = i - 1  
    return i >= 0
```

cherche si  $x$  existe dans la liste  $L$ . Elle renvoie `True` si c'est le cas et `False` sinon.

### 2) Entiers $\leftrightarrow$ liste ... (6 points)

- a) La fonction **int\_to\_list(n,p)** qui convertit le nombre  $n$  (entier naturel à au plus  $p$  chiffres) en une liste de ses chiffres éventuellement complétés par des 0 pour atteindre  $p$  chiffres, peut s'écrire de la manière suivante :

```
def int_to_list(n, p):  
    L = []  
    while n != 0:  
        L.append(n % 10)  
        n = n // 10  
    for i in range(p-len(L)):  
        L.append(0)  
    return L
```

- b) La fonction **list\_to\_ints(L)** qui, à partir de la liste  $L$  non vide, ne contenant que des chiffres (de 0 à 9), retourne le couple (*left*, *right*), avec :

- *left* le nombre construit avec les chiffres de  $L$  lus de gauche à droite,
- *right* le nombre construit avec les chiffres de  $L$  lus de droite à gauche.

peut s'écrire de la manière suivante :

```
def list_to_int(L):  
    left = 0  
    right = 0  
    n = len(L)  
    for i in range(n):  
        left = left * 10 + L[i]  
        right = right * 10 + L[n-i-1]  
    return (left, right)
```

### 3) Histogramme et tri ... (4 points)

- a) La fonction **hist(L)** qui retourne un histogramme (sous la forme d'une liste) des chiffres présents dans la liste **L**, peut s'écrire de la manière suivante :

```
def hist(L):  
    """  
    L contient uniquement des chiffres (de 0 à 9)  
    hist(L) construit un histogramme des valeurs de L  
    """  
    H = [0] * 10 # ou boucle avec append...  
    for e in L:  
        H[e] += 1  
    return H
```

- b) La fonction **sort(L)** , utilisant la fonction **hist**, qui trie la liste **L** en ordre croissant (la nouvelle liste construite devant être retournée), peut s'écrire de la manière suivante :

```
def sort(L):  
    H = hist(L)  
    L = []  
    for i in range(10):  
        for nb in range(H[i]):  
            L.append(i)  
    return L
```

#### 4) Kaprekar ... (5 points)

La fonction de **Kaprekar** utilise toutes les fonctions précédemment créées et peut s'écrire de la manière suivante :

```
def Kaprekar(n, p):  
    L = []  
    while not test(n, L):  
        L.append(n)  
        digits = int_to_list(n, p)  
        digits = sort(digits)  
        (low, high) = list_to_int(digits)  
        n = high - low  
        L.append(n)  
    return L
```