Algorithmique
Correction Contrôle n° 2 (C2)

INFO-SUP S2# – EPITA

novembre 2019

Solution 1 **(Un peu de cours... – *4 points*)**

1. C'est l'arbre B représenté figure 1.

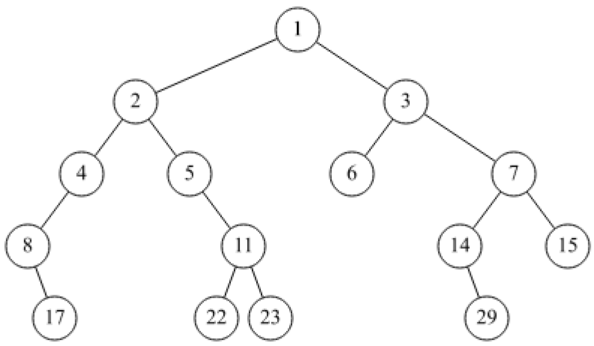

FIGURE 1 – Arbre binaire

2. La longueur de cheminement interne de l'arbre B est : $17 = 0+1+1+2+2+2+3+3+3$
3. La profondeur moyenne externe de l'arbre B est : $21/6 = 3,5$ ($lce = 21 = 4+4+4+2+4+3$)

Solution 2 **(ABR : chemin de recherche – *2 points*)**

Les séquences ② et ④ sont impossibles :

① 50 - 15 - 48 - 22 - 46 - 42
50, on descend à gauche - 15, on descend à droite - 48 on descend à gauche - 22, on descend à droite - 46, on descend à gauche - **42**

② 48 - 15 - 45 - 22 - 47 - 42
48, on descend à gauche - 15, on descend à droite - **45, on descend à gauche** - 22, on descend à droite - **47 ne peut se trouver là, il n'est pas inférieur à 45 !**

③ 15 - 22 - 45 - 43 - 35 - 42
15, on descend à droite - 22, on descend à droite - 45, on descend à gauche - 43, on descend à gauche - 35, on descend à droite - **42**

④ 22 - 45 - 43 - 15 - 35 - 42
22, on descend à droite - 45, on descend à gauche - 43, on descend à gauche - **15 ne peut pas se trouver là, il n'est pas supérieur à 22**

Solution 3 **(Transposée - *3 points*)**

Spécifications :

La fonction `transpose(`A`)` construit et retourne la matrice transposée de la matrice non vide A.

```python
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
```

Solution 4 **(Symétrie verticale – *5 points*)**

Spécifications :

La fonction `v_symmetric(`M`)` vérifie si la matrice M est symétrique selon un axe horizontale (symétrie verticale).

```python
def v_symmetric(M):
    (l, c) = (len(M), len(M[0]))
    ldiv2 = l // 2
    i = 0
    test = True
    while i < ldiv2 and test:
        j = 0
        while j < c and test:
            test = M[i][j] == M[l-i-1][j]
            j += 1
        i += 1
    return test

def v_symmetric2(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
```

Solution 5 **(Maximum Path Sum – *2 points*)**

Spécifications :

La fonction `maxpath(`B`)` retourne la plus grande valeur des branches de l'arbre binaire B (0 si l'arbre est vide)

```python
def maxpath(B):
    if B == None:
        return 0
    else:
        return B.key + max(maxpath(B.left), maxpath(B.right))
```

Solution 6 **(Full ? –** ***3 points*****)**

Corrections ci-dessous : Directement adaptées des fonctions qui testent si un arbre est dégénéré !

```python
# not the most optimized (to many test)
def full0(T):
    if T == None :  # this test might be in a call function!
        return True
    elif T.left == None or T.right == None: #single point
        return False
    else :
        return full0(T.left) and full0(T.right)

# the optimized version (only 2 tests each time)
def __full(B):
    '''
    B not empty
    '''
    if B.left == None:
        if B.right == None:
            return True
        else:
            return False
    else:
        if B.right == None:
            return False
        else:
            return __full(B.left) and full(B.right)

def full(B):
    return B == None or __full(B)

# a nice version
def __full2(B):
    '''
    B not empty
    '''
    leftEmpty = (B.left == None)
    if B.right == None:
        return leftEmpty
    else:
        return not leftEmpty and __full2(B.left) and __full2(B.right)

def full2(B):
    return B == None or __full2(B)
```

Solution 7 **(Mystery –** ***2 points*****)**

```python
>>> what(B)
[[5], [2, 12], [-1, 0, 4, 1], [4, 11, -2], [15]]
```