Algorithmique
Correction Contrôle n° 2 (C2)

INFO-SUP S2 – EPITA

4 mars 2019 - 9 : 00

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

1. Mesures :
   (a) La taille de l'arbre `B` est : `11`
   (b) La hauteur de l'arbre `B` est : `4`
   (c) La longueur de cheminement de l'arbre `B` est : `27`
   (d) La profondeur moyenne externe de l'arbre `B` est : `15/4 = 3.75`
2. En utilisant la numérotation hiérarchique, les nœuds de l'arbre `B` sont :
   1, 2, 3, 5, 7, 10, 11, 14, 21, 28, 29

Solution 2 **(Maximum Gap –** ***4 points*****)**

Spécifications :
La fonction `maxgap(`M`)` retourne le gap maximum des lignes de la matrice non vide M.

```python
def gaplist(L):
    """
    returns the gap of the list L (not empty)
    """
    valMin = L[0]
    valMax = L[0]
    for i in range(1, len(L)):
        valMin = min(valMin, L[i])
        valMax = max(valMax, L[i])
    return valMax - valMin

def maxgap(M):
    mgap = gaplist(M[0])
    for i in range(1, len(M)):
        mgap = max(mgap, gaplist(M[i]))
    return mgap
```

In one function (gaplist inlined) :

```python
            def maxgap2(M):
                mgap = 0
                (l, c) = (len(M), len(M[0]))
                for i in range(l):
                    valMin = M[i][0]
                    valMax = M[i][0]
                    for j in range(1, c):
                        valMin = min(valMin, M[i][j])
                        valMax = max(valMax, M[i][j])
                    mgap = max(mgap, valMax - valMin)
                return mgap
```

Solution 3 **(Recherche – *4 points*)**

Spécifications :

La fonction `searchMatrix(M, x)` retourne la position (i, j) de la première valeur `x` trouvée dans la matrice `M` (non vide) ou $(-1, -1)$ si `x` $\notin$ `M`.

```
                    def searchMatrix(M, x):

                        (i, lin, col) = (0, len(M), len(M[0]))
                        found = -1

                        while i < lin and found == -1:
                            j = 0
                            while j < col and M[i][j] != x:
                                j += 1
                            if j != col:
                                found = j
                            i += 1

                        if found != -1:
                            return (i-1, found)
                        else:
                            return (-1, -1)
```

Solution 4 **(Tests – *4 points*)**

Spécifications : La fonction `equal(`$B1$`,` $B2$`)` vérifie si les arbres $B1$ et $B2$ sont indentiques.

```
        def equal(B1, B2):
            if B1 == None:
                return B2 == None
            elif B2 == None:
                return False
            elif B1.key == B2.key:
                return equal(B1.left, B2.left) and equal(B1.right, B2.right)
            else:
                return False

#-------------------------------------------------------------

      def equal2(B1, B2):
          if B1 == None or B2 == None:
              return B1 == B2
          else:
              return (B1.key == B2.key) \
                     and equal2(B1.left, B2.left) \
                     and equal2(B1.right, B2.right)
```

Solution 5 **(Feuilles – *2 points*)**

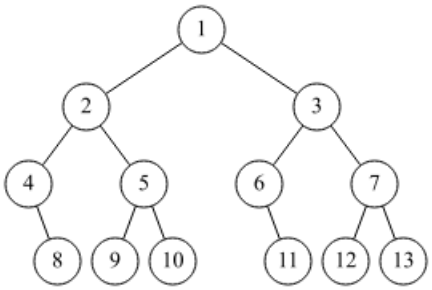
Spécifications :

La fonction `leaves(`B`)` calcule le nombre de feuilles de l'arbre binaire B.

```
1        def leaves(B):
2            if B == None:
3                return 0
4            else:
5                if B.left == B.right:    # B.left == None and B.right == None
6                    return 1
7                else:
8                    return leaves(B.left) + leaves(B.right)
```

Solution 6 **(Mystery – *3 points*)**

1. Arbre binaire résultat de l'application `mystery([4, 8, 2, 9, 5, 10, 1, 6, 11, 3, 12, 7, 13])` :



2. (a) *L'arbre est un ABR si la liste est* triée en ordre croissant.

   (b) *L'arbre est complet si la liste est* de taille $= 2^h - 1$ avec h un entier naturel.