Algorithmique
Correction Contrôle n° 2 (C2)

Info-sup S2 – Epita

5 mars 2018 - 8 : 30

Solution 1 **(Un peu de cours... –** ***5 points*****)**

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

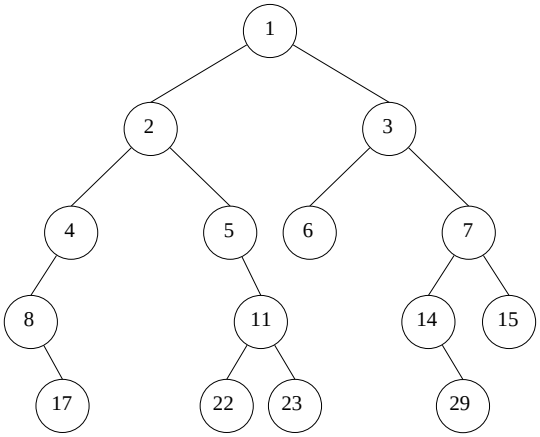

Figure 1 – Arbre binaire

2. Les noeuds externes *(en ordre hiérarchique)* de l'arbre `B` sont : `6, 15, 17, 22, 23, 29`
3. La longueur de cheminement externe de l'arbre `B` est : `21`
4. La profondeur moyenne interne de l'arbre `B` est : `17/9` soit environ `1,89 (1.888888...)`
5. La particularité d'un arbre localement complet est de n'être composé que de points doubles et de feuilles.

Solution 2 **(Arbre Binaire : Ordres –** ***2 points*****)**

1. L'arbre B correspondant à ces deux séquences est le suivant :

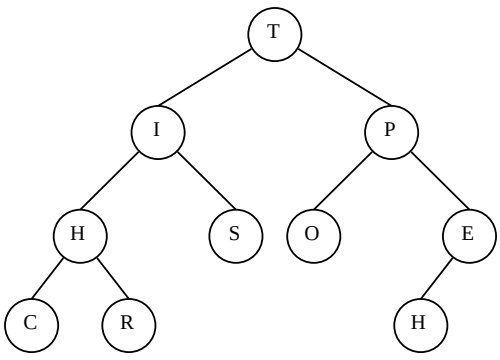

Figure 2 – Arbre binaire B

2. Les valeurs *préfixe* de rencontre de l'arbre B sont : T I H C R S P O E H

Solution 3 **(Symétrie – *5 points*)**

Spécifications :

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

```
            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 4 **(Test implémentation hiérarchique – *5 points*)**

Spécifications :

La fonction `object_vs_list(`B`,` L`)` vérifie si les deux arbres B, en représentation classique ("objet"), et L, en *implémentation hiérarchique*, sont identiques.

```
            def object_vs_list(B, L, i=1):
                if B == None or i >= len(L) or L[i] == None:
                    return B == None and (i >= len(L) or L[i] == None)
                elif B.key != L[i]:
                    return False
                else:
                    return object_vs_list(B.left, L, 2*i)
                          and object_vs_list(B.right, L, 2*i+1)
```

Solution 5 **(Père et fils –** ***4 points*****)**

Spécifications :

La fonction `copywithparent` construit à partir de l'arbre binaire "classique" B (`BinTree`) un arbre binaire équivalent (contenant les mêmes valeurs aux même places) mais avec le père renseigné en chaque nœud (`BinTreeParent`).

```
1              # first version: parent is pass as parameter
2
3              def __copy(B, p):
4                  if B == None:
5                      return None
6                  else:
7                      C = BinTreeParent(B.key, p, None, None)
8                      C.left = __copy(B.left, C)
9                      C.right = __copy(B.right, C)
10                     return C
11
12             def copywithparent(B):
13                 return __copy(B, None)
14
15 #-------------------------------------------------------------------------
16             # second version: parent is set going up
17
18             def __copy2(B):
19                 '''
20                 B is not None
21                 '''
22                 C = BinTreeParent(B.key, None, None, None)
23                 if B.left != None:
24                     C.left = __copy2(B.left)
25                     C.left.parent = C
26                 if B.right != None:
27                     C.right = __copy2(B.right)
28                     C.right.parent = C
29                 return C
30
31             def copyWithParent2(B):
32                 if B == None:
33                     return None
34                 else:
35                     return __copy2(B)
```