Algorithmique
Correction Contrôle n° 2 (C2)

Info-sup (S2#) – Epita

8 novembre 2017 - 8 : 30

Solution 1 **(Représentations et questions ... –** ***4 points*****)**

1. Elle s'appelle la bijection premier fils-frère droit.
2. La représentation de l'arbre général de la figure 1 selon cette représentation binaire est celui de la figure 2.

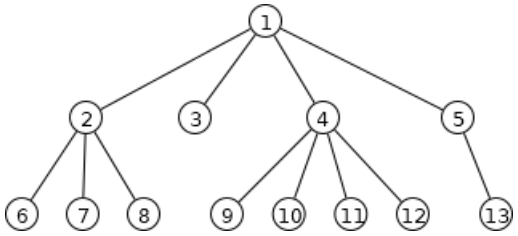


Figure 1 – Arbre général

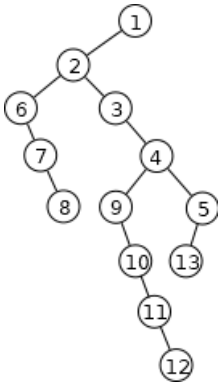


Figure 2 – Arbre binaire représentant l'arbre général

3. C'est une bijection, il y a autant de noeuds dans l'arbre binaire qu'il y en a dans l'arbre général qu'il représente. Il suffit donc de faire un parcours de l'arbre binaire en comptant les noeuds rencontrés.
4. Seuls les fils sont plus haut que leur père. Les frères eux sont à la même hauteur. Il suffit donc de faire un parcours de l'arbre binaire en initialisant la hauteur et la hauteur maximum à 0. On augmente cette hauteur de 1 uniquement lorsqu'on suit un lien premier fils. A chaque fois qu'elle augment, on la compare à la hauteur maximum qui prend sa valeur si celle-ci est supérieure. A la fin du parcours, hauteur maximum est égal à la hauteur de l'arbre général.
5. Cette recherche est qualifiée de positive.

Solution 2 **(ABR : insertions – *3 points*)**

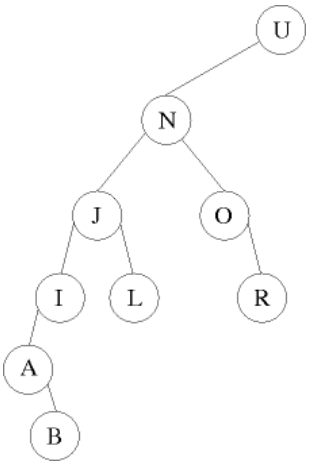


FIGURE 3 – Un joli ABR, construit en feuille

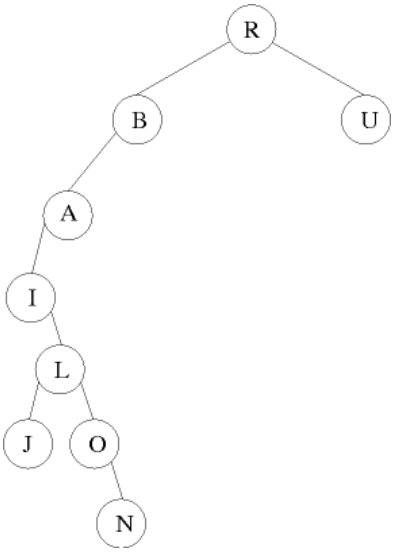


FIGURE 4 – Un joli ABR, construit en racine

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 **(Symmetric – *4 points*)**

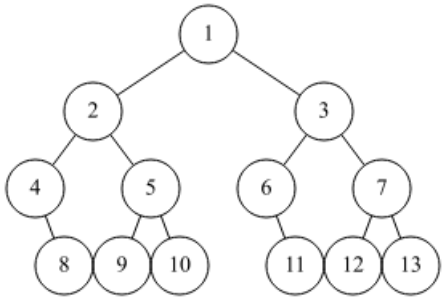
```
        def __symmetric(B1, B2):
            if B1 == None or B2 == None:
                return B1 == B2
            else:
                if B1.key != B2.key:
                    return False
                else:
                    return __symmetric(B1.left, B2.right) \
                    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

        def symmetric(B):
            return B == None or __symmetric(B.left, B.right)
```

Solution 5 **(Maximum Path Sum – *3 points*)**

```
1    def maxpath(B):
2        if B == None:
3            return 0
4        else:
5            return B.key + max(maxpath(B.left), maxpath(B.right))
```

Solution 6 **(Mystery – *2 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 décroissant.

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