# Algorithmique
# Correction Partiel n° 2 (P2)

INFO-SUP (S2) – EPITA

*29 mai 2017 - 13h45*

***Solution 1*** **(Arbres 234 ... –** ***6 points*****)**

1. Les insertions successives des valeurs $\{Q, U, E, S, T, I, O, N, B, A, Z, Y, K\}$, donnent l'arbre 2.3.4. de la figure 1.

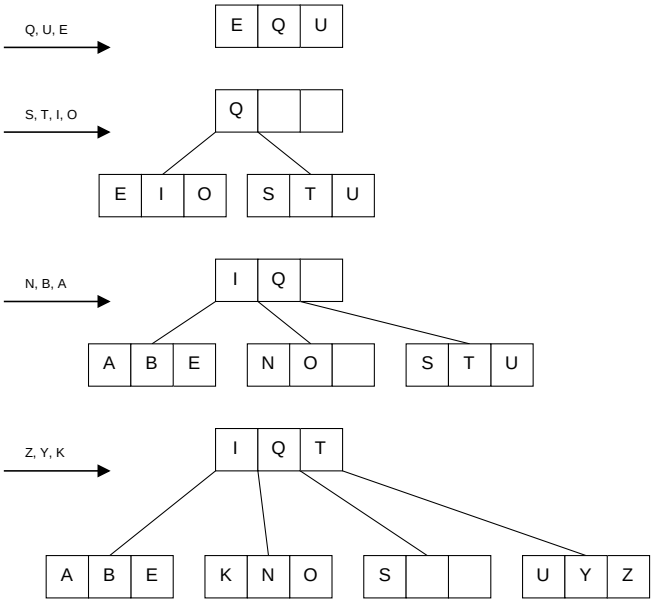



FIGURE 1 – Arbre 2.3.4. après les insertions des valeurs $\{Q, U, E, S, T, I, O, N, B, A, Z, Y, K\}$.

2. L'arbre rouge-noir associé à l'abre 2.3.4. de la question précédente est l'arbre de la figure 2.

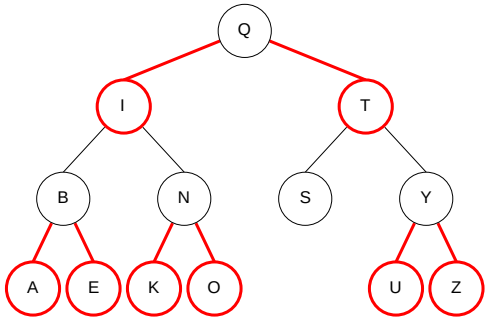


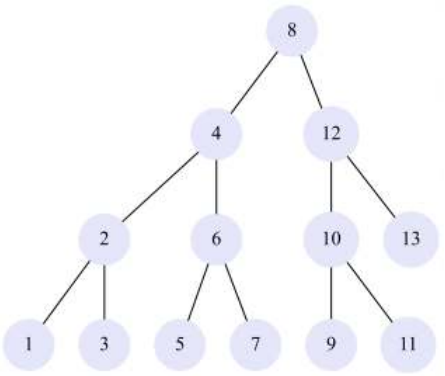

FIGURE 2 – Arbre rouge-noir associé à l'arbre 2.3.4. de la figure 1.

3. Trois propriétés d'un arbre 2.3.4 pourraient être :
   — Un arbre 2.3.4. est un arbre de recherche,
   — Les noeuds d'un arbre 2.3.4. sont de trois types : 2-noeud, 3-noeud ou 4-noeud,
   — Tous les noeuds externes (feuilles) d'un arbre 2.3.4. sont au même niveau,
   — Un arbre 2.3.4. est équilibré.
4. Trois propriétés d'un arbre rouge-noir pourraient être :
   — Un arbre rouge-noir est un arbre binaire de recherche,
   — Les noeuds d'un arbre rouge-noir sont soit rouge, soit noir,
   — La racine d'un arbre rouge-noir est toujours noire,
   — Dans un arbre rouge-noir, un noeud fils qui contient un élément jumeau de celui de son père est rouge,
   — Dans un arbre rouge-noir, les branches ont un nombre de liens (noeuds) noirs égal à la hauteur de l'arbre 2.3.4. correspondant,
   — Un arbre rouge-noir est équilibré.
5. *Méthode **simple** permettant de déterminer la taille d'un arbre 2.3.4. en utilisant l'arbre bicolore qui le représente :* Compter tous les noeuds noirs de l'arbre bicolore.

---

***Solution 2* (Arbres et mystère – *3 points*)**

1. *Arbre construit par* `makeTree(13)` *:*




2. *Propriétés de l'arbre construit par* `makeTree(n)` *($n > 0$)* :
   (a) Arbre parfait
   (b) Arbre binaire de recherche

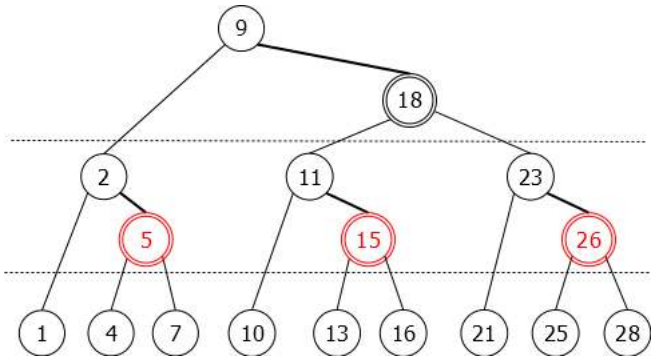
***Solution 3* (ABR → AVL – *5 points*)**

**Spécifications :**
La fonction `madeAVLfromBST(`$B$`)` contruit une copie de l'arbre binaire $B$ avec les déséquilibres renseignés en chaque nœud.

```
            def BST2AVL(B):
                if B == None:
                    return(None, -1)

                else:
                    A = avl.AVL(B.key, None, None, 0)
                    (A.left, hl) = BST2AVL(B.left)
                    (A.right, hr) = BST2AVL(B.right)

                    A.bal = hl - hr
                    return (A, 1 + max(hl, hr))


            def MakeAVL(B):
                (A, h) = BST2AVL(B)
                return A
```

***Solution 4* (Arbres AA – *6 points*)**

1. *Arbre AA obtenu après insertion de la valeur 4 dans l'arbre de la figure 6.*




2. **Spécifications :**
La fonction `insertAA(`$x$`, `$A$`)` insère $x$ dans l'arbre AA $A$ sauf si celui-ci est déjà présent. Elle retourne l'arbre résultat de l'insertion.

```
def insertAA(x, A):
    if A == None:
        return AAtree(x, None, None, 1)

    else:
        if x < A.key:
            A.left = insertAA(x, A.left)
            if A.left.level == A.level:
                A = skew(A)
                if A.right.right != None and A.right.right.level == A.level:
                    A = split(A)

        elif x > A.key:
            A.right = insertAA(x, A.right)
            if A.right.right != None and A.right.right.level == A.level:
                A = split(A)

        return A
```