Algorithmique
Correction Partiel n° 2 (P2)

INFO-SUP S2# – EPITA

8 Jan. 2018 - 9 : 00

Solution 1 **(La taille en plus –** ***4 points*****)**

```python
        def __addSize(B):
            if B == None:
                return(None, 0)
            else:
                C = BinTreeSize(B.key, None, None, 1)
                (C.left, size1) = __addSize(B.left)
                (C.right, size2) = __addSize(B.right)
                C.size += size1 + size2
                return (C, C.size)

# another version

        def addSize2(B):
            if B == None:
                return(None, 0)
            else:
                (left, size1) = addSize2(B.left)
                (right, size2) = addSize2(B.right)
                size = 1 + size1 + size2
                return (BinTreeSize(B.key, left, right, size), size)
```

```python
        def copyWithSize(B):
            (C, size) = addSize(B)
            return C
```

Solution 2 **(Ajout avec mise à jour de la taille –** ***3 points*****)**

Spécifications :

La fonction `addwithsize(`B`, `x`)`, ajoute x en feuille dans l'arbre binaire de recherche B (`BinTreeSize()`).

```python
            def addBSTSize(x, A):
                if A == None:
                    A = BinTreeSize(x, None, None, 1)
                else:
                    if x <= A.key:
                        A.left = addBSTSize(x, A.left)
                    else:
                        A.right = addBSTSize(x, A.right)
                    A.size += 1
                return A
```

Solution 3 **(Médian –** ***6 points*****)**

1. B ABR de n éléments dont le $k^{ème}$ élément ($1 \leq k \leq n$) se trouve en racine :
   — taille(g(B)) $= k - 1$
   — taille(d(B)) $= n - k$

2. **Spécifications :**
   La fonction `nthBST(`B`, `k`)` avec B un ABR non vide et $1 \leq k \leq taille(B)$, retourne l'arbre dont la racine contient le $k^{ème}$ élément de B .

```python
def nthBST(B, k):

    if B.left == None:
        leftSize = 0
    else:
        leftSize = B.left.size

    if leftSize == k - 1:
        return B
    elif k <= leftSize:
        return nthBST(B.left, k)
    else:
        return nthBST(B.right, k - leftSize - 1)


def nthBST2(B, k):

    if B.left == None:
        if k == 1:
            return B
        else:
            return nthBST2(B.right, k - 1)

    else:
        if k == B.left.size + 1:
            return B
        elif k <= B.left.size:
            return nthBST2(B.left, k)
        else:
            return nthBST2(B.right, k - B.left.size - 1)
```

Spécifications :
La fonction `median(`B`)` retourne la valeur médiane de l'ABR B s'il est non vide, la valeur `None` sinon.

```python
def median(B):
    if B != None:
        return nthBST(B, (B.size+1) // 2).key
    else:
        return None
```

Solution 4 **(What is this ? –** ***3 points*****)**

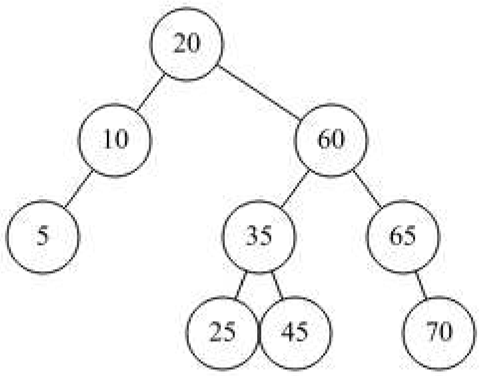
1. *Résultats pour*
   (a) `test(`B_2`)` : True
   (b) `test(`B_3`)` : False5

2. `test(`B`)` vérifie si l'arbre binaire B est h-équilibré.

3. Pour optimiser cette fonction : si le booléen du premier appel est faux, il est possible d'éviter le deuxième en retournant directement `(?, False)`.

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

AVL résultat depuis la liste $[25, 60, 35, 10, 20, 5, 70, 65, 45]$.



Solution 6 **(Arbres 2-3-4 –** ***3 points*****)**

1. Avec éclatement à la descente :
   Pour cette méthode, nous avons pour l'ajout des valeurs 4, 11, 9 et 18 les arbres successifs des figures 1, 2, 3 et 4.

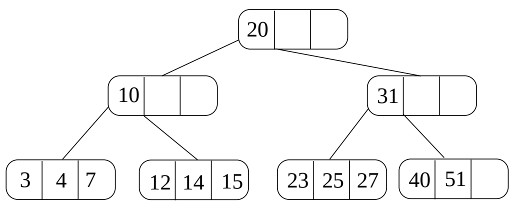


Figure 1 – Insertion de 4 avec éclatement à la descente

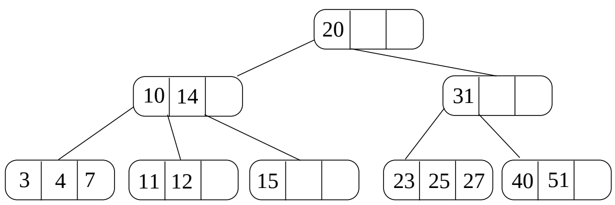


Figure 2 – Insertion de 11 avec éclatement à la descente

2. Représentation bicolore de l'arbre234 :
   En considérant les 3-nœuds penchés à droite, on obtient l'arbre bicolore de la figures 5.

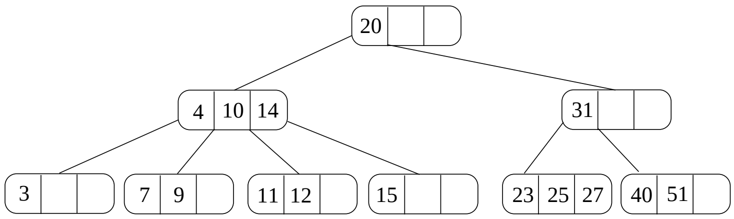


Figure 3 – Insertion de 9 avec éclatement à la descente

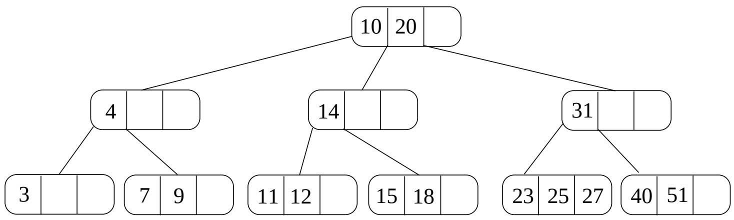


Figure 4 – Insertion de 18 avec éclatement à la descente

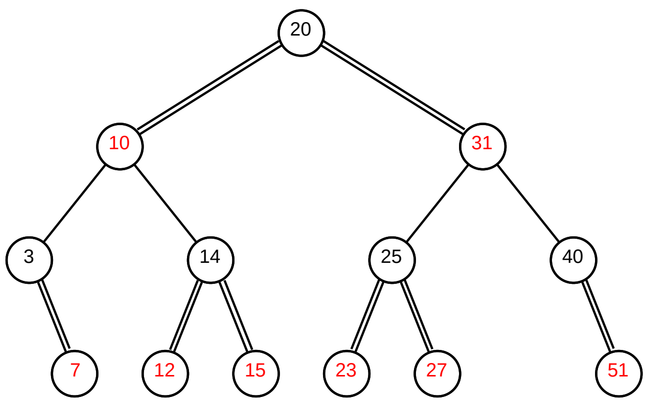


Figure 5 – Arbre bicolore