

Algorithmique

Correction Partiel n° 2 (P2)

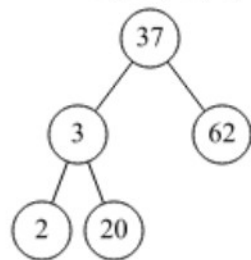
INFO-SUP S2# – EPITA

18 janvier 2021 - 8h30-10h30

Solution 1 (AVL – 3 points)

AVL résultat depuis la liste [20, 62, 37, 3, 2, 6, 74, 73, 14].

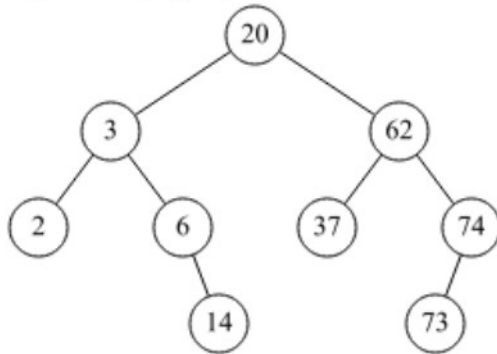
Arbre créé par insertions de 20, 62, 37, 3, 2 :



Rotations :

rlr(20) rdg(20)
rr(20) rd(20)

Arbre après ajout de 6, 74, 73, 14 :



Rotations :

rlr(37) rgd(37)
lr(37) rg(37)

Solution 2 (Arbre 2.3.4 → Arbre bicolore – 2 points)

1. Arbre bicolore correspondant à l'arbre 2.3.4 du sujet :
Les nœuds "doubles" sont rouges !

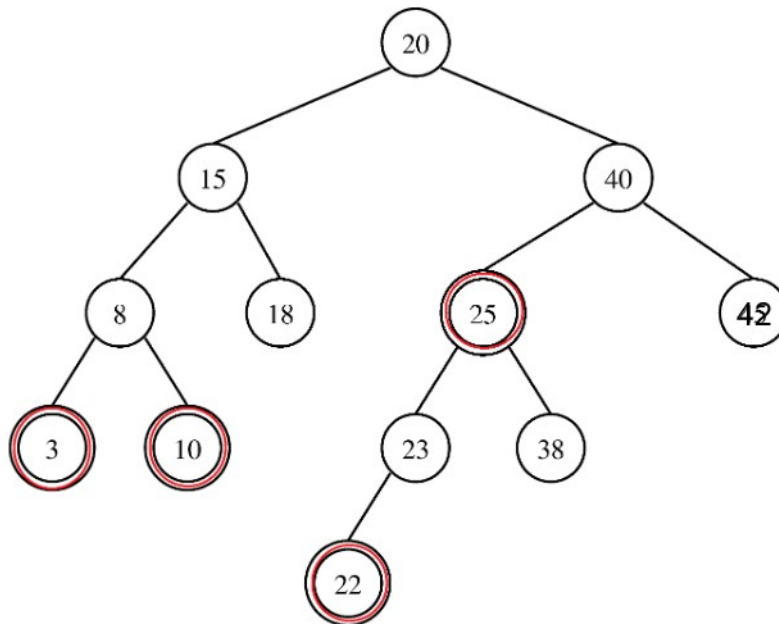


FIGURE 1 – Arbre bicolore

2. L'arbre obtenu n'est pas un AVL.
Le nœud contenant 40 a un déséquilibre de 2 (l'arbre n'est pas h-équilibré).

Solution 3 (Symmetric – 4 points)

Spécifications :

La fonction `symmetric(B)` vérifie si l'arbre S est *symétrique*.

Fonction supplémentaire :

`__symmetric(A, B)` vérifie si A est l'arbre binaire symétrique de l'arbre binaire B .

```
1 def __symmetric(B1, B2):
2     if B1 == None or B2 == None:
3         return B1 == B2
4     else:
5         if B1.key != B2.key:
6             return False
7         else:
8             return __symmetric(B1.left, B2.right) \
9                 and __symmetric(B1.right, B2.left)
10
11 def symmetric(B):
12     return B == None or __symmetric(B.left, B.right)
```

Solution 4 (ABR : Insertion en feuille – 4 points)

Spécifications :

La fonction `leaf_insert(B, x)` ajoute l'élément x dans l'arbre binaire de recherche B , dans une nouvelle feuille, sauf si x est déjà présent. Elle retourne un couple (l'arbre, un booléen indiquant si l'insertion a eu lieu).

```
1     def insertBST(x, B):
2
3         if B == None:
4             return (BinTree(x, None, None), True)
5
6         else:
7             if x == B.key:
8                 ins = False
9             elif x < B.key:
10                (B.left, ins) = insertBST(x, B.left)
11            else:
12                (B.right, ins) = insertBST(x, B.right)
13
14            return (B, ins)
```

Solution 5 (Average Balance Factor – 5 points)

Spécifications :

La fonction `average_balances(B)` calcule la moyenne des déséquilibres de tous les nœuds de l'arbre binaire (class `BinTree`) B .

Fonction supplémentaire :

La fonction `__sum_balances(B)` retourne un triplet (sum_bal , $height$, $size$) avec

- sum_bal : la somme des déséquilibres des nbœuds de B
- $height$: la hauteur de B
- $size$: la taille de B

```
1     def __sum_balances(B):
2         """
3         return (sum balance factors, height, size)
4         """
5         if B == None:
6             return (0, -1, 0)
7         else:
8             (sl, hl, nl) = __sum_balances(B.left)
9             (sr, hr, nr) = __sum_balances(B.right)
10            return (sl + sr + (hl - hr), 1 + max(hl, hr), 1 + nr + nl)
11
12     def average_balances(B):
13         if B == None:
14             return 0
15         else:
16             (sum_bal, height, size) = __sum_balances(B)
17             return (sum_bal / size)
```

Solution 6 (What is this? – 2 points)

Arbre résultat de $\text{New}(B)$:

