

# Trabajo Practico N°2

Nombre: Victor Ramirez

## Ejercicio 1

Crear un modulo de nombre **avltree.py** Implementar las siguientes funciones:

**rotateLeft(Tree, avlnode)**

**Descripción:** Implementa la operación rotación a la izquierda

**Entrada:** Un Tree junto a un AVLnode sobre el cual se va a operar la rotación a la izquierda

**Salida:** retorna la nueva raíz

**rotateRight(Tree, avlnode)**

**Descripción:** Implementa la operación rotación a la derecha

**Entrada:** Un Tree junto a un AVLnode sobre el cual se va a operar la rotación a la derecha

**Salida:** retorna la nueva raíz

```
12
13 def rotateLeft(Tree , avlnode):
14
15     raiz_vieja = avlnode
16     Tree.root = avlnode.rightrightnode
17     raiz_nueva = Tree.root
18     aux = raiz_nueva.leftnode
19     raiz_nueva.leftnode = raiz_vieja
20     raiz_nueva.parent = raiz_vieja.parent
21
22     if aux != None:
23         raiz_vieja.rightrightnode = aux
24         raiz_vieja.parent = raiz_nueva
25         aux.parent = raiz_vieja
26     return raiz_nueva
27
28
29 def rotateRight(Tree , avlnode):
30     raiz_vieja = avlnode
31     Tree.root = avlnode.leftnode
32     raiz_nueva = Tree.root
33     aux = Tree.root.rightrightnode
34     raiz_nueva.rightrightnode = raiz_vieja
35     raiz_nueva.parent = raiz_vieja.parent
36
37     if aux != None:
38         raiz_vieja.leftnode = aux
39         raiz_vieja.parent = raiz_nueva
40         aux.parent = raiz_vieja
41
42     return raiz_nueva
43
```

## Ejercicio 2

Implementar una función recursiva que calcule el elemento balanceFactor de cada subárbol siguiendo la siguiente especificación:

### **calculateBalance(AVLTree)**

**Descripción:** Calcula el factor de balanceo de un árbol binario de búsqueda.

**Entrada:** El árbol AVL sobre el cual se quiere operar.

---

ICUYO - Facultad de Ingeniería.  
Licenciatura en Ciencias de la Computación.

Algoritmos y Estructuras de Datos II:  
Árboles Balanceados: AVL

---

**Salida:** El árbol AVL con el valor de balanceFactor para cada subarbol

```
45 #Recibe la raiz de un arbol
46 def calculateBalance(ALVTree):
47     node = ALVTree.root
48     if ALVTree == None:
49         return
50     #Queremos actualizar el node.bf
51     height_left = 0
52     height_right = 0
53     if node.leftnode != None:
54         height_left = calculateBalance_balanceRecursive(node.leftnode)
55     if node.rightnode != None:
56         height_right = calculateBalance_balanceRecursive(node.rightnode)
57     bf = height_left - height_right
58     node.bf = bf
59
60     return node
61
62
63 def calculateBalance_balanceRecursive(node):
64
65     if node == None:
66         return 0
67
68     height_left = calculateBalance_balanceRecursive(node.leftnode)
69     height_right = calculateBalance_balanceRecursive(node.rightnode)
70     #print(f"Altura izquierda {height_left} , Altura derecha {height_right}")
71
72     bf = height_left - height_right
73     node.bf = bf
74
75     altura = max(height_left , height_right )
76
77     return 1 + altura
78
```

## Ejercicio 3

Implementar una función en el módulo `avltree.py` de acuerdo a las siguientes especificaciones:

### `reBalance(AVLTree)`

**Descripción:** balancea un árbol binario de búsqueda. Para esto se deberá primero calcular el **balanceFactor** del árbol y luego en función de esto aplicar la estrategia de rotación que corresponda.

**Entrada:** El árbol binario de tipo AVL sobre el cual se quiere operar.

**Salida:** Un árbol binario de búsqueda balanceado. Es decir luego de esta operación se cumple que la altura (h) de su subárbol derecho e izquierdo difieren a lo sumo en una unidad.

```
def reBalance(AVLTree):
    calculateBalance(AVLTree)
    reBalance_recursive(AVLTree , AVLTree.root)
    return AVLTree
```

```
def reBalance_recursive(AVLTree , node):

    if node.leftnode != None:
        reBalance_recursive(AVLTree, node.leftnode)

    if node.bf < -1:
        if node.rightnode.bf > 0:
            AVLTree.root.rightnode = rotateRight(AVLTree.root.rightnode, node.rightnode)
            rotateLeft(AVLTree,node)
            calculateBalance(AVLTree)
        else:
            rotateLeft(AVLTree,node)
            calculateBalance(AVLTree)

    elif node.bf > 1:
        if node.leftnode.bf < 0:
            AVLTree.root.leftnode = rotateLeft(AVLTree.root.leftnode,node.leftnode)
            rotateRight(AVLTree,node)
            calculateBalance(AVLTree)
        else:
            rotateRight(AVLTree,node)
            calculateBalance(AVLTree)

    if node.rightnode != None:
        reBalance_recursive(AVLTree , node.rightnode)
```

## Ejercicio 4:

Implementar la operación `insert()` en el módulo `avltree.py` garantizando que el árbol binario resultante sea un árbol AVL.

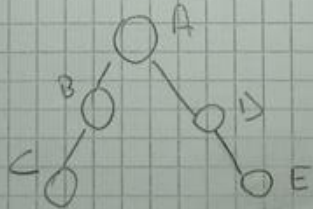
```
120
121 def insert(AVLTree , key):
122     new_node = AVLNode()
123     new_node.key = key
124     current = AVLTree.root
125
126     if current != None:
127         return insertR(current,new_node)
128     else:
129         AVLTree.root = new_node
130
131     return reBalance(AVLTree)
132
133
134 def insertR(current,newNode):
135
136
137     if newNode.key < current.key:
138         if current.leftnode == None:
139             current.leftnode = newNode
140             newNode.parent = current
141             return newNode
142         else:
143             return insertR(current.leftnode,newNode)
144
145     elif newNode.key > current.key:
146         if current.rightnode == None:
147             current.rightnode = newNode
148             newNode.parent = current
149             return newNode
150         else:
151             return insertR(current.rightnode,newNode)
152
153     else:
154         return None
```

## Ejercicio 6:

1. Responder V o F y justificar su respuesta:
  - a. \_\_\_\_ En un AVL el penúltimo nivel tiene que estar completo
  - b. \_\_\_\_ Un AVL donde todos los nodos tengan factor de balance 0 es completo
  - c. \_\_\_\_ En la inserción en un AVL, si al actualizarle el factor de balance al padre del nodo insertado éste no se desbalanceó, entonces no hay que seguir verificando hacia arriba porque no hay cambios en los factores de balance.
  - d. \_\_\_\_ En todo AVL existe al menos un nodo con factor de balance 0.

A)

a) Suponemos un árbol AVL no completo, entonces:  
Contraejemplo:



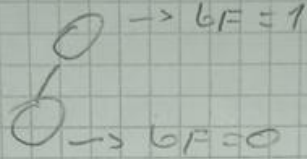
```
graph TD; A((A)) --- B((B)); A --- D((D)); B --- C((C)); D --- E((E))
```

No necesariamente tiene que estar completo

B)

b) Por contraejemplo, suponemos un árbol AVL no completo, entonces va a existir un nodo tal que:

Falso



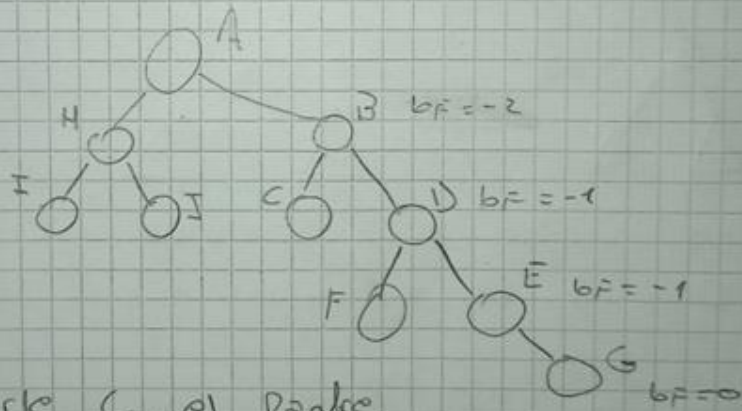
```
graph TD; P(( )) --> C(( ))
```

Por lo tanto, para que todos sus nodos tengan  $bF = 0$  tiene que ser completo.

c)

c) Falso, contra ejemplo?

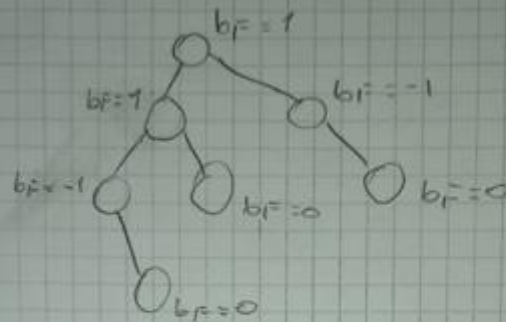
Si tengo



Al insertarle G, el padre  
está balanceado, sin embargo tengo que  
seguir verificando hacia arriba ya que  
el nodo B quedó desbalanceado.

d)

d) Falso, contra ejemplo?



Si no tomamos en cuenta las hojas, podemos  
ver que es un árbol AVL y ningún nodo  
tiene  $bf=0$ .

## Ejercicio 7:

Sean  $A$  y  $B$  dos AVL de  $m$  y  $n$  nodos respectivamente y sea  $x$  un key cualquiera de forma tal que para todo key  $a \in A$  y para todo key  $b \in B$  se cumple que  $a < x < b$ . Plantear un algoritmo  $O(\log n + \log m)$  que devuelva un AVL que contenga los key de  $A$ , el key  $x$  y los key de  $B$ .

### Ejercicio 7



y se cumple que  $a < x < b$  tal que  $a$  y  $b$  son keys de los subárboles  $A$  y  $B$  respectivamente.

- ① Comparar las alturas de  $A$  y  $B$ .
- ② Insertar  $x$  en el árbol con mayor altura a la altura del árbol con menor altura.
- ③ El subárbol izq de  $x$  la raíz. Va a ser el árbol con menor altura.
- ④ El subárbol derecho de  $x$  es el subárbol izq de  $B$ .
- ⑤ El subárbol derecho de  $B$  queda igual.
- ⑥ Se balancea desde  $x$  hacia la raíz.

- Op 1: Altura de  $B$   $O(\log n)$
- Op 2: Altura de  $A$   $O(\log m)$
- Op 3: Insertar  $x$  a la altura de  $A$   $O(\log m)$
- Op 4: Insertar  $A$  al lado izq de  $x$   $O(1)$
- Op 5: Asigna al lado derecho de  $x$  el subárbol izq de  $B$   $O(1)$
- Op 6: Rebalancea desde  $x$  hacia la raíz  $O(\log n)$

Ent:  $3 \log m + \log n$

$O(\log m + \log n)$