

## Trabajo Práctico Árboles balanceados AVL

### 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

```
14 def rotateLeft(Tree, avlnode):
15     # El nuevo nodo raiz es el hijo derecho de la antigua raiz
16     newRootNode = avlnode.rightrightnode
17     avlnode.rightrightnode = None
18     # Si el nodo raiz anterior era la raiz del arbol, asignar su hijo derecho como nueva raiz del arbol
19     if avlnode.parent == None:
20         Tree.root = newRootNode
21         newRootNode.parent = None
22     else:
23         # Sino asigno como padre del nuevo nodo raiz al padre del antiguo nodo raiz
24         newRootNode.parent = avlnode.parent
25     # Si el hijo derecho de la antigua raiz tenia un hijo izquierdo, este pasa a ser hijo derecho de la antigua raiz
26     if newRootNode.leftnode != None:
27         avlnode.rightrightnode = newRootNode.leftnode
28         newRootNode.leftnode.parent = avlnode
29     # El antiguo nodo raiz pasa a ser el hijo izquierdo del nuevo nodo raiz
30     newRootNode.leftnode = avlnode
31     avlnode.parent = newRootNode
32     # Retorno la nueva raiz del arbol
33     return newRootNode
```

```
35 def rotateRight(Tree,avlnode):
36     # El nuevo nodo raiz es el hijo izquierdo de la antigua raiz
37     newRootNode = avlnode.leftnode
38     avlnode.leftnode = None
39     # Si el nodo raiz anterior era la raiz del arbol, asignar su hijo izquierdo como nueva raiz del arbol
40     if avlnode.parent == None:
41         Tree.root = newRootNode
42         newRootNode.parent = None
43     else:
44         # Sino asigno como padre del nuevo nodo raiz al padre del antiguo nodo raiz
45         newRootNode.parent = avlnode.parent
46     # Si el hijo izquierdo de la antigua raiz tenia un hijo derecho, este pasa a ser hijo izquierdo de la antigua raiz
47     if newRootNode.rightrightnode != None:
48         avlnode.leftnode = newRootNode.rightrightnode
49         newRootNode.rightrightnode.parent = avlnode
50     # El antiguo nodo raiz pasa a ser el hijo derecho del nuevo nodo raiz
51     newRootNode.rightrightnode = avlnode
52     avlnode.parent = newRootNode
53     # Retorno la nueva raiz del arbol
54     return newRootNode
```

## 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.

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

```
58 def findHeight(node):
59     # Caso base
60     if node == None:
61         return -1
62     # Casos recursivos
63     leftHeight = findHeight(node.leftnode)
64     rightHeight = findHeight(node.rightnode)
65     # Comparo las alturas y retorno recursivamente la mayor entre la izq y la der
66     if leftHeight >= rightHeight:
67         return 1 + leftHeight
68     return 1 + rightHeight
69
70 def calcBalanceR(avlnode):
71     # Caso base
72     if avlnode == None:
73         return 0
74     # Casos recursivos
75     leftHeight = calcBalanceR(avlnode.leftnode)
76     rightHeight = calcBalanceR(avlnode.rightnode)
77     # Calculo y actualizo el balance factor del nodo
78     avlnode.bf = leftHeight - rightHeight
79     # El retorno recursivo lleva implicito el valor de la altura del nodo mediante una funcion recursiva que calcula la altura
80     return 1 + findHeight(avlnode)
81
82
83 def calculateBalance(AVLTree):
84     if AVLTree.root == None:
85         return
86     # Llamado a funcion recursiva
87     calcBalanceR(AVLTree.root)
88     return AVLTree
```

## 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.

```
92 def reBalanceR(AVLTree, avlnode):
93     # Caso base
94     if avlnode == None:
95         return
96     # Detecto si el nodo tiene desbalance hacia la derecha
97     if avlnode.bf < -1:
98         # Se detecta si el hijo derecho tiene un hijo izquierdo
99         if avlnode.rightrightnode.bf > 0:
100             rotateRight(AVLTree, avlnode.rightrightnode)
101             avlnode = rotateLeft(AVLTree, avlnode)
102         else:
103             avlnode = rotateLeft(AVLTree, avlnode)
104         calculateBalance(AVLTree)
105     # Detecto si el nodo tiene desbalance hacia la izquierda
106     elif avlnode.bf > 1:
107         # Se detecta si el hijo izquierdo tiene un hijo derecho
108         if avlnode.leftleftnode.bf < 0:
109             rotateLeft(AVLTree, avlnode.leftleftnode)
110             avlnode = rotateRight(AVLTree, avlnode)
111         else:
112             avlnode = rotateRight(AVLTree, avlnode)
113         calculateBalance(AVLTree)
114     # Llamadas recursivas
115     reBalanceR(AVLTree, avlnode.leftnode)
116     reBalanceR(AVLTree, avlnode.rightrightnode)
117
118 def reBalance(AVLTree):
119     if AVLTree.root == None:
120         return
121     calculateBalance(AVLTree)
122     reBalanceR(AVLTree, AVLTree.root)
123     return AVLTree
```

### Ejercicio 4:

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

### Ejercicio 5:

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

## Parte 2

### 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.