Alumno: Manuel Matías QUESADA RICCIERI

Parte 1

Ejercicio 1

rotateLeft(Tree,avlnode)

```
def rotateLeft(AVLTree, AVLNode):
    newNode = AVLNode.rightnode
    AVLNode.rightnode = newNode.leftnode
    newNode.leftnode = AVLNode

    newNode.parent = AVLNode.parent
    AVLNode.parent = newNode
    if AVLNode.rightnode != None:
        AVLNode.rightnode.parent = AVLNode

    AVLNode.bf = getBalanceFactor(AVLNode)
    newNode.bf = getBalanceFactor(newNode)

    return newNode
```

rotateRight(Tree,avlnode)

```
def rotateRight(AVLTree, AVLNode):
    newNode = AVLNode.leftnode
    AVLNode.leftnode = newNode.rightnode
    newNode.rightnode = AVLNode

    newNode.parent = AVLNode.parent
    AVLNode.parent = newNode
    if AVLNode.leftnode != None:
        AVLNode.leftnode.parent = AVLNode

    AVLNode.bf = getBalanceFactor(AVLNode)
    newNode.bf = getBalanceFactor(newNode)

    return newNode
```

Funciones Implementadas:

```
def getBalanceFactor(node):
    leftHeight = 0
    rightHeight = 0

    if node.leftnode != None:
        leftHeight = treeHeight(node.leftnode)

    if node.rightnode != None:
        rightHeight = treeHeight(node.rightnode)

    return leftHeight - rightHeight
```

Ejercicio 2

`calculateBalance(AVLTree)`

```
def calculateBalance(AVLTree):
    return calculateBalanceR(AVLTree, AVLTree.root)

def calculateBalanceR(AVLTree, AVLNode):
    if AVLNode == None:
        return None

    leftheight = treeHeight(AVLNode.leftnode)
    rightheight = treeHeight(AVLNode.rightnode)

    AVLNode.bf = leftheight - rightheight

    calculateBalanceR(AVLTree, AVLNode.leftnode)
    calculateBalanceR(AVLTree, AVLNode.rightnode)

    return AVLTree
```

`Funciones Implementadas:`

```
def treeHeight(currentNode):
    if currentNode == None:
        return 0
    else:
        Left = treeHeight(currentNode.leftnode)
        Right = treeHeight(currentNode.rightnode)

        if Left > Right:
            return Left + 1
        else:
            return Right + 1
```

Ejercicio 3

`reBalance(AVLTree)`

```
def reBalance(AVLTree):
    if AVLTree.root != None:
        AVLTree.root = reBalanceR(AVLTree, AVLTree.root)

    if checkRebalance(AVLTree, AVLTree.root) == True:
        AVLTree.root = reBalanceR(AVLTree, AVLTree.root)

def reBalanceR(AVLTree, AVLNode):

    if AVLNode.leftnode != None:
        AVLNode.leftnode = reBalanceR(AVLTree, AVLNode.leftnode)
    if AVLNode.rightnode != None:
        AVLNode.rightnode = reBalanceR(AVLTree, AVLNode.rightnode)

    AVLNode.bf = getBalanceFactor(AVLNode)

    if AVLNode.bf > 1:
        if AVLNode.leftnode.bf > 0:
            AVLNode = rotateRight(AVLTree, AVLNode)
        else:
            AVLNode.leftnode = rotateLeft(AVLTree, AVLNode.leftnode)
            AVLNode = rotateRight(AVLTree, AVLNode)

    elif AVLNode.bf < -1:
        if AVLNode.rightnode.bf < 0:
            AVLNode = rotateLeft(AVLTree, AVLNode)
        else:
            AVLNode.rightnode = rotateRight(AVLTree, AVLNode.rightnode)
            AVLNode = rotateLeft(AVLTree, AVLNode)

    return AVLNode
```

`Funciones Implementadas:`

```
def checkRebalance(AVLTree, AVLNode):
    calculateBalance(AVLTree)

    if AVLNode == None:
        return None

    balanceFactor = AVLNode.bf
    if balanceFactor != [-1, 0, 1]:
        return True

    left = checkRebalance(AVLTree, AVLNode.leftnode)
    if left != None:
        return left
    right = checkRebalance(AVLTree, AVLNode.rightnode)
    if right != None:
        return right

    return None
```

Ejercicio 4:

`insert()`

```
def insert(AVLTree, element, key):
    newNode = AVLNode()
    newNode.value = element
    newNode.key = key

    if AVLTree.root == None:
        newNode.bf = 0
        AVLTree.root = newNode
        return key
    else:
        key = insertR(newNode, AVLTree.root)

    calculateBalance(AVLTree)

    if key != None:
        reBalance(AVLTree)
    return key
```

```
def insertR(newNode, currentNode):
    if newNode.key > currentNode.key:
        if currentNode.rightnode == None:
            currentNode.rightnode = newNode
            newNode.parent = currentNode
            return newNode.key
        currentNode = currentNode.rightnode
        return insertR(newNode, currentNode)
    elif newNode.key < currentNode.key:
        if currentNode.leftnode == None:
            currentNode.leftnode = newNode
            newNode.parent = currentNode
            return newNode.key
        currentNode = currentNode.leftnode
        return insertR(newNode, currentNode)
    else:
        return None
```

Ejercicio 5:

`delete()`

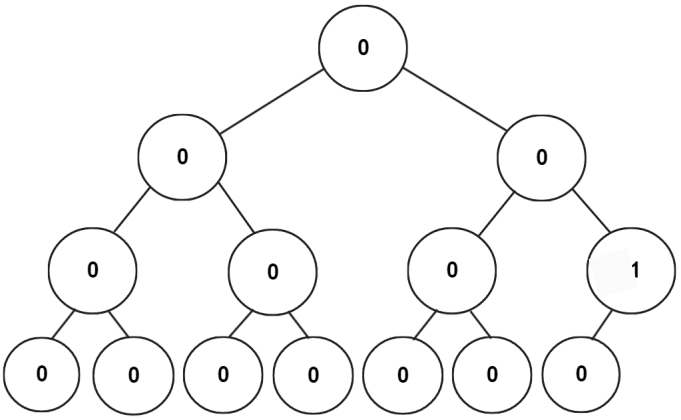
```
def delete(AVLTree, element):
    node = searchR(AVLTree.root, element)
    if node == None:
        return
    else:
        key = deleteR(AVLTree, node)

    calculateBalance(AVLTree)

    if key != None:
        reBalance(AVLTree)
    return key
```

```
def deleteR(B, AVLNode):
    if AVLNode.rightnode == None:
        if AVLNode.leftnode == None:
            #eliminar hijo
            if AVLNode.parent.leftnode != None and AVLNode.parent.leftnode == AVLNode:
                AVLNode.parent.leftnode = None
                return AVLNode.key
            elif AVLNode.parent.rightnode != None and AVLNode.parent.rightnode == AVLNode:
                AVLNode.parent.rightnode = None
                return AVLNode.key

        #hijo del lado izquierdo
        if AVLNode.parent.leftnode != None and AVLNode.parent.leftnode == AVLNode:
            AVLNode.parent.leftnode = AVLNode.leftnode
            return AVLNode.key
        elif AVLNode.parent.rightnode != None and AVLNode.parent.rightnode == AVLNode:
            AVLNode.parent.rightnode = AVLNode.leftnode
            return AVLNode.key
    else:

        #hijo del lado derecho
        if AVLNode.leftnode == None:
            if AVLNode.parent.leftnode == AVLNode:
                AVLNode.parent.leftnode = AVLNode.rightnode
                return AVLNode.key
            elif AVLNode.parent.rightnode == AVLNode:
                AVLNode.parent.rightnode = AVLNode.rightnode
                return AVLNode.key

        else:
            #2 hijos, elimina el menor de los mayores
            newNode = smaller(AVLNode.rightnode)
            AVLNode.value = newNode.value
            oldKey = AVLNode.key
            AVLNode.key = newNode.key

            if newNode.parent.leftnode == newNode:
                newNode.parent.leftnode = None
            elif newNode.parent.rightnode == newNode:
                newNode.parent.rightnode = None
            return oldKey

def smaller(node):
    if node.leftnode != None:
        current = smaller(node.leftnode)
        if current != None:
            return current
    else: return node
```

Parte 2

Ejercicio 6:

1. Responder V o F y justificar su respuesta:
    a. **_F_** En un AVL el penúltimo nivel tiene que estar completo



Suponemos que es verdadero que en un AVL el penúltimo nivel tiene que estar completo, así que construimos un árbol AVL que demuestre lo contrario

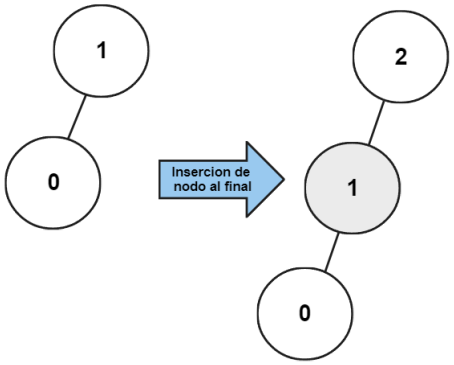
En este árbol AVL podemos ver como el penúltimo nivel no está completo, y podemos comprobarlo calculando el factor de balance de cada uno de sus nodos.

Por ello, a través de una contradicción podemos decir que existe al menos un ejemplo que demuestra que la afirmación es falsa

    b. **_V_** Un AVL donde todos los nodos tengan factor de balance 0 es completo

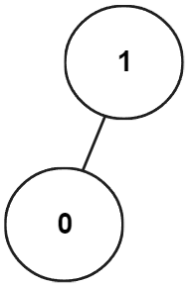
Suponemos que la afirmación es falsa, entonces consideramos que existe un árbol AVL que no sea completo y donde el balance de factor de todos sus nodos sean iguales a 0. Al no ser completo entendemos que hay un nodo que tiene un único hijo, pero al hacer el cálculo, su balance de factor se vuelve en 1 o -1, contradiciendo la hipótesis.

    c. **_F_** 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.



En el ejemplo donde tenemos una raíz con un hijo, e insertamos un nodo conectado al hijo, el balance de factor de su padre se convierte en 1 y no se desbalancea, pero si hace falta realizar una verificación, ya que la raíz si se encuentra desbalanceada (bf = 2), y en ese caso si se debe hacer una rotación.

d. **_F_** En todo *AVL* existe al menos un nodo con factor de balance 0.



Si no contamos las hojas es falso, ya que existe el ejemplo en el que existe un árbol balanceado en el que su raíz tenga un balance de factor 1 y su hijo 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 ∈ A y para todo key b ∈ 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.

Este es el algoritmos que yo plantearía para devolver un árbol AVL que contenga los keys de A, B y x

1. Creo el nodo key = x

2. Comparo las alturas de los árboles A y B, esto nos daría un orden de complejidad O(log n) para calcular las keys de A y O(log m) para las de B

3. Ahora pueden ocurrir 3 casos, para cada caso el nodo de x irá conectado a los nodos para construir un nuevo árbol.
   a) Si la altura del árbol B es mayor a la del árbol A: La raíz del árbol B será la raíz del árbol AVL, a su derecha irán todos las keys mayores y a la izquierda estará conectado al nodo con key = x, los menores a B se encontrarán a la derecha de x, y a su izquierda conectará con la raíz del árbol A completo.

   b) Si la altura del árbol A es mayor a la del árbol B: La raíz del árbol A será la raíz del árbol AVL, a su izquierda irán todas las keys menores y a la derecha estará conectado al nodo con key = x, los mayores de A se encontrarán a la izquierda de x, y a su derecha conectará con la raíz del árbol B completo

   c) Si la altura de A y B son iguales: Si son iguales el nodo de key = x pasará a ser la raíz del árbol, a su izquierda estará conectada a la raíz del árbol A y a su derecha estará conectada a la raíz del árbol B

4. Retorno el árbol completo

Ejercicio 8:

Considere una rama truncada en un AVL como un camino simple desde la raíz hacia un nodo que tenga una referencia None (que le falte algún hijo). Demuestre que la mínima longitud (cantidad de aristas) que puede tener una rama truncada en un AVL de altura h es h/2 (tomando la parte entera por abajo).

Cualquier camino desde la raíz hasta un nodo que no esté completo puede ser una rama truncada según la definición del ejercicio. Dicho nodo puede no ser necesariamente un nodo hoja.

Consideramos una rama truncada que comienza en la raíz del árbol y termina en un nodo que le falta un hijo. Si la altura del árbol es h, entonces la rama truncada tiene como máximo de longitud h-1, en el caso de que el hijo faltante sea una hoja.

Debido a que la rama truncada termina en un nodo con un hijo faltante, se puede deducir que todos los nodos internos de la rama tienen ambos hijos exceptuando a la del hijo faltante. Entonces si la longitud de la rama truncada es h-1 el nodo que tiene el hijo faltante está en el nivel h-2, y el subárbol opuesto a la dirección del hijo faltante tiene altura al menos de h/2, debido a la propiedad de balanceo de los AVL.

Si el nodo está en el nivel h-2 entonces el subárbol opuesto a la dirección del hijo faltante tiene altura al menos h/2-1, ya que todos los nodos internos en ese subárbol tienen ambos hijos

En cualquiera de los casos, la altura mínima que puede tener la rama truncada es h/2, ya que ese es el menor valor posible de la altura del subárbol que contiene el nodo con el hijo faltante