Árboles Balanceados: AVL

Mauro José Sorbello Diaz Legajo: 14006

Parte 1

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

```python
def rotateLeft(Tree, avlnode):
  newnode = avlnode.rightnode
  #NOTE: Si la nueva raiz tiene un hijo a la izquierda, ese hijo pasa a ser un hijo derecho del nodo rotado, cambiamos el parent tambien
  avlnode.rightnode = newnode.leftnode
  if newnode.rightnode != None:
    newnode.leftnode.parent = avlnode
  newnode.parent = avlnode.parent
  #NOTE: Vemos el parent de avlnode
  if avlnode.parent == None:
    #NOTE: Significa que avlnode era la raiz, por ende newnode sera la raiz
    Tree.root = newnode
  else:
    if avlnode.parent.leftnode == avlnode:
      #NOTE: Ahora el padre de newnode es el padre de avlnode
      avlnode.parent.leftnode = newnode
    else:
      avlnode.parent.rightnode = newnode
  #NOTE: Hacemos que avlnode sea el hijo izquierdo de newnode
  newnode.leftnode = avlnode
  #NOTE: Hacemos que newnode sea el padre de avlnode
  avlnode.parent = newnode
  return Tree.root
```

```python
def rotateRight(Tree, avlnode):
  newnode = avlnode.leftnode
  #NOTE: Si la nueva raiz tiene un hijo a la derecha, ese hijo pasa a ser un hijo izquierdo del nodo rotado, cambiamos el parent tambien
  avlnode.leftnode = newnode.rightnode
  if newnode.rightnode != None:
    newnode.rightnode.parent = avlnode
  newnode.parent = avlnode.parent
  #NOTE: Vemos el parent de avlnode
  if avlnode.parent == None:
    #NOTE: Significa que avlnode era la raiz, por ende newnode sera la raiz
    Tree.root = newnode
  else:
    if avlnode.parent.leftnode == avlnode:
      #NOTE: Ahora el padre de newnode es el padre de avlnode
      newnode.parent = avlnode.parent
    else:
      newnode.parent = avlnode.parent
  #NOTE: Hacemos que avlnode sea el hijo derecho de newnode
  newnode.rightnode = avlnode
  #NOTE: Hacemos que newnode sea el padre de avlnode
  avlnode.parent = newnode
  return Tree.root
```

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

```python
def auxCalcBalance(AVLTree, current):
  hleft = 0
  currentleft = current
  currentright = current
  while currentleft != None:
    hleft += 1
    currentleft = currentleft.leftnode
  hright = 0
  while currentright != None:
    hright += 1
    currentright = currentright.rightnode
  balance = hleft - hright
  current.bf = balance
  if current.leftnode != None:
    auxCalcBalance(AVLTree, current.leftnode)
  if current.rightnode != None:
    auxCalcBalance(AVLTree, current.rightnode)


def calculateBalance(AVLTree):
  currentNode = AVLTree.root
  if (currentNode != None):
    auxCalcBalance(AVLTree, currentNode)
```

Ejercicio 3

Implementar una funcion en el modulo avltree.py de acuerdo a las siguientes especifcaciones:

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.

```python
def reBalanceAux(AVLTree, node):
    #CASOS CON NODO.BF = 2
    if node.bf < -1:
        #CASO A Y B
        if (node.rightnode.bf == -1) or (node.rightnode.bf == 0):
            rotateLeft(AVLTree, node)
        #CASO C
        if node.rightnode.bf == 1:
            if node.leftnode == None:
                rotateLeft(AVLTree, node)
                rotateRight(AVLTree, node.rightnode)
            else:
                rotateLeft(AVLTree, node)
        #CASO D Y E
    if node.bf > 1:
        if (node.leftnode.bf == 1) or (node.leftnode.bf == 0):
            rotateRight(AVLTree, node)
        #CASO F
        if (node.leftnode.bf == -1):
            if node.rightnode == None:
                rotateRight(AVLTree, node)
                rotateLeft(AVLTree, node.leftnode)
            else:
                rotateRight(AVLTree, node)
    #RECALCULO EL BALANCE
    calculateBalance(AVLTree)
    #SI ESTA BALANCEADO
    if abs(node.bf) <= 1:
        if node.leftnode != None:
            reBalanceAux(AVLTree, node.leftnode)
        if node.rightnode != None:
            reBalanceAux(AVLTree, node.leftnode)


def reBalance(AVLTree):
    reBalanceAux(AVLTree, AVLTree.root)
    return AVLTree
```

Ejercicio 4:

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

```python
def addNode(AVLTree, current, newNode):
    if current.key > newNode.key:
        if current.leftnode == None:
            current.leftnode = newNode
            newNode.parent = current
            calculateBalance(AVLTree)
            return reBalance(AVLTree)
        else:
            addNode(AVLTree, current.leftnode, newNode)
    else:
        if current.rightnode == None:
            current.rightnode = newNode
            newNode.parent = current
            calculateBalance(AVLTree)
            return reBalance(AVLTree)
        else:
            addNode(AVLTree, current.rightnode, newNode)

def balanceNodeUp(AVLTree, current):
    if abs(current.bf) > 1:
        return reBalanceAux(AVLTree, current)
    else:
        if current.parent != None:
            return balanceNodeUp(AVLTree, current.parent)

def insert(AVLTree, element, key):
    newNode = AVLNode()
    newNode.key = key
    newNode.value = element
    if AVLTree.root != None:
        addNode(AVLTree, AVLTree.root, newNode)
    else:
        AVLTree.root = newNode
    return AVLTree
```

Ejercicio 5:

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

```python
def searchD(current, element):
    if current == None:
        return None
    if current.value == element:
        return current
    else:
        None
    #Derecha
    if current.value < element:
        if searchD(current.rightnode, element) != None:
            return searchD(current.rightnode, element)
    elif current.value > element:
        #Izquierda
        if searchD(current.leftnode, element) != None:
            return searchD(current.leftnode, element)


def search(AVLTree, element):
    node = searchD(AVLTree.root, element)
    if node != None:
        return node.key

#AHORA SI PASAMOS AL DELETE

def menor(AVLTree, current):
    if current != None:
        men = menor(AVLTree, current.leftnode)
        if men != None:
            return men
        else:
            return current
```

```python
def deleteNode(AVLTree, node):
    #CASO 1: ELIMINAR AVLTree.ROOT
    #CASO 1.1: EL ROOT A ELIMINAR TIENE UN HIJO IZQUIERDA
  if node == AVLTree.root:
    if (node.leftnode != None):
      if node.rightnode == None:
        if (node.leftnode != None) and (node.leftnode.parent == node):
          node = node.leftnode
        else:
          if (node.rightnode != None) and (node.rightnode.parent == node):
            node = node.leftnode
    #CASO 1.2: EL ROOT A ELIMINAR TIENE UN HIJO DERECHA
    else:
      if node.rightnode != None:
        if node.leftnode.parent == node:
          node = node.rightnode
        else:
          if (node.rightnode.parent == node):
            node = node.rightnode
    #CASO 2: ELIMINAR UNA HOJA
  if (node.rightnode == None):
    if (node.leftnode == None):
      if node.parent.leftnode != None:
        if node == node.parent.leftnode:
          node.parent.leftnode = None
        else:
          if node.parent.rightnode != None:
            node.parent.rightnode = None
    #CASO 3.1: EL NODO A ELIMINAR TIENE UN HIJO IZQUIERDA
  if (node.leftnode != None):
    if node.rightnode == None:
      if (node.parent.leftnode != None) and (node.parent.leftnode == node):
        node.parent.leftnode = node.leftnode
      else:
        if (node.parent.rightnode != None) and (node.parent.rightnode == node):
          node.parent.rightnode = node.leftnode
    #CASO 3.2: EL NODO A ELIMINAR TIENE UN HIJO DERECHA
  else:
    if node.rightnode != None:
      if node.parent.leftnode == node:
        node.parent.leftnode = node.rightnode
      else:
        if (node.parent.rightnode == node):
          node.parent.rightnode = node.rightnode
```

```python
    #CASO 4: EL NODO A ELIMINAR TIENE 2 O MAS HIJOS
  if (node.leftnode != None) and (node.rightnode != None):
    menorMayor = menor(AVLTree, node.rightnode)
    node.key = menorMayor.key
    node.value = menorMayor.value
    if menorMayor.parent.leftnode == menorMayor:
      menorMayor.parent.leftnode = None
    if (menorMayor.rightnode == None) and (node.rightnode == menorMayor):
      node.rightnode = None
    if menorMayor.leftnode == None and (node.leftnode == menorMayor):
      node.leftnode = None

  #CUALQUIERA DE LOS SEIS CASOS DE REBALANCEO PUEDE DARSE, Y EL DESBALANCE PUEDE PROPAGARSE HACIA NODOS SUPERIORES, POR ENDE HAY QUE ANALIZAR EL ARBOL ENTERO, NO COMO EN INSERCION QUE ES SOLO EL CAMINO HACIA LA RAIZ.
  reBalance(AVLTree)

def delete(AVLTree, element):
    #Busco si existe el nodo con la funcion auxiliar de search
    node = searchD(AVLTree.root, element)
    if node != None:
        return deleteNode(AVLTree, node)
```

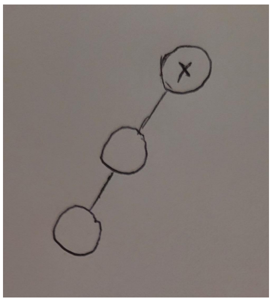
Parte 2

Ejercicio 6:

1. Responder V o F y justificar su respuesta:

a. _V_ En un AVL el penúltimo nivel tiene que estar completo

DEMOSTRACIÓN:
Supongo que hay un penúltimo nivel incompleto
Entonces: Existe un nodo x que es antepenúltimo y solo tiene hijos en la izquierda.



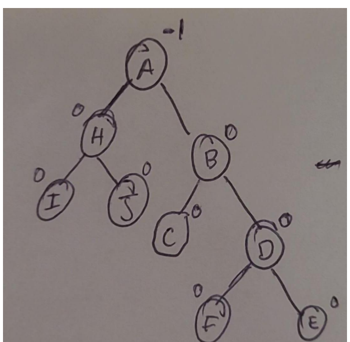
SU BF = 2

Entonces no cumple con ser un AVL, por ende es Falso.
Como el contrario al enunciado es Falso, entonces en un AVL el penúltimo nivel debe estar completo

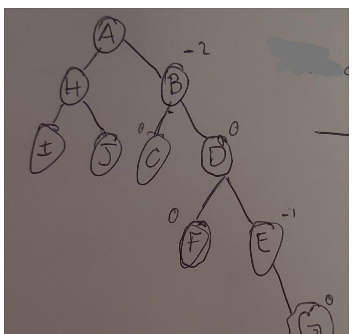
b. _V_ Un AVL donde todos los nodos tengan factor de balance 0 es completo
c.
DEMOSTRACIÓN
Supongo que existe un AVL donde todos los nodos tienen bf = 0 y no es completo.
Entonces existe un nodo del árbol que solo tiene un hijo (definicion de incompleto)
Si tiene un solo hijo su bf sera -1 ó 1, por ende hay una contradicción y es Falso.
Como lo contrario al enunciado es falso, el enunciado es Verdadero

d. _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.

CONTRAEJEMPLO:
TENGO EL AVL:

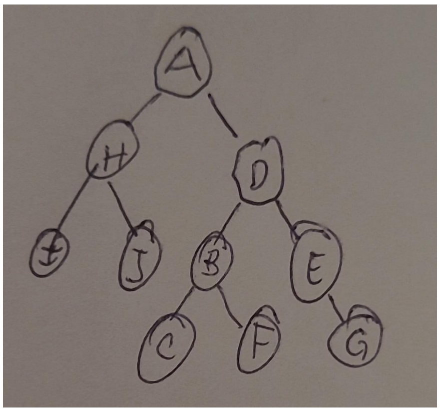


INSERTO G:



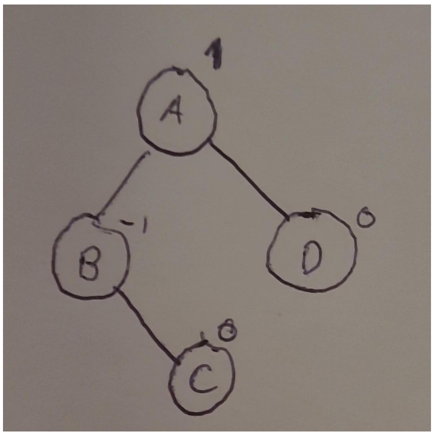
Observamos que E no se desbalancea ya que su BF = .-1, pero B si se desbalanceó, BF = -2

Entonces para que cumpla la condición de AVL deberemos balancear nuevamente.



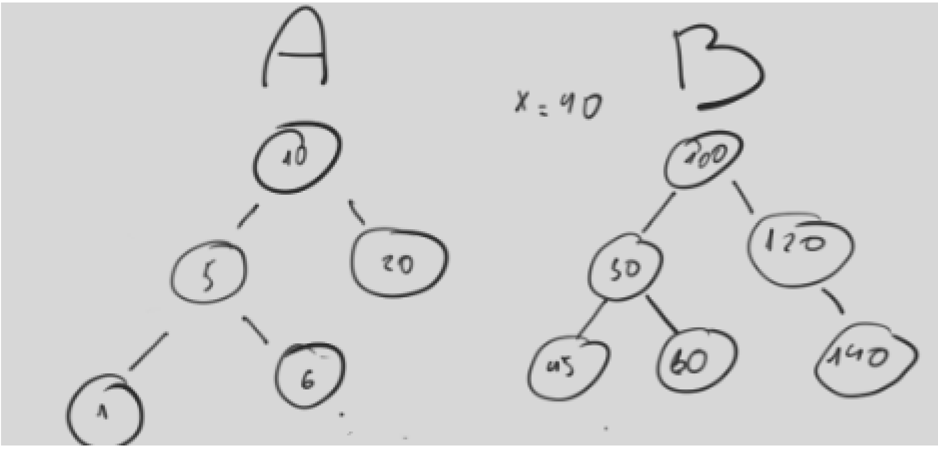
e. _F (Sin considerar las hojas)_ En todo *AVL* existe al menos un nodo con factor de balance 0.

CONTRAEJEMPLO:



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.



RESPUESTA:

Primero calculo la altura de A.

Después busco el nodo de B que esté a la misma altura de A.

Luego sabemos que A < x < B, por ende insertamos a x, como parent del nodo de B (que está a la misma altura que la altura de A)

Y hacemos que A sea el hijo izquierdo de x.

Por último deberemos balancear desde x, hasta B.root