NOMBRE: ALVARO BLANCO
LEGAJO: 10622
TRABAJO PRACTICO Arbol AVL

PARTE I

```
from  linkedlist import add, printList
from classes import *


def rotateLeft(Tree, avlnode):
  if avlnode and avlnode.rightnode:
    newRoot = avlnode.rightnode
    avlnode.rightnode = newRoot.leftnode

    if newRoot.leftnode != None:
      newRoot.leftnode.parent = avlnode
    newRoot.parent = avlnode.parent

    if avlnode.parent == None:
      Tree.root = newRoot
    else:
      if avlnode.parent.leftnode == avlnode:
        avlnode.parent.leftnode = newRoot
      else:
        avlnode.parent.rightnode = newRoot

    newRoot.leftnode = avlnode
    avlnode.parent = newRoot


def rotateRight(Tree, avlnode):
  if avlnode and avlnode.leftnode:
    newRoot = avlnode.leftnode
    avlnode.leftnode = newRoot.rightnode

    if newRoot.rightnode != None:
      newRoot.rightnode.parent = avlnode
    newRoot.parent = avlnode.parent

    if avlnode.parent == None:
      Tree.root = newRoot
    else:
      if avlnode.parent.rightnode == avlnode:
        avlnode.parent.rightnode = newRoot
      else:
        avlnode.parent.leftnode = newRoot

    newRoot.rightnode = avlnode
    avlnode.parent = newRoot
```

```
def height(root):
  if root == None:
    return -1

  lh = height(root.leftnode)
  rh = height(root.rightnode)

  return max(lh, rh) + 1


def calculateBalance(AVLTree):
  calculateBf(AVLTree.root)
  return AVLTree


def calculateBf(root):
  left = height(root.leftnode)
  right = height(root.rightnode)
  root.bf = left - right
  if root.leftnode != None:
    calculateBf(root.leftnode)
  if root.rightnode != None:
    calculateBf(root.rightnode)


def insert(B, element, key):
  root = B.root
  newNode = AVLNode()
  newNode.value = element
  newNode.key = key
  if root == None:
    B.root = newNode
    return key
  else:
    newKey = insertNode(newNode, root)
    return newKey


def insertAVL(B, element, key):
  root = B.root
  newNode = AVLNode()
  newNode.value = element
  newNode.key = key
  if root == None:
    B.root = newNode
    return key
  else:
    newKey = insertNode(newNode, root)
  reBalance(B)
  return newKey


def insertNode(newNode, current):
  if newNode.key < current.key:
    if current.leftnode == None:
      current.leftnode = newNode
      newNode.parent = current
```

```
    else:
      insertNode(newNode, current.leftnode)
  elif newNode.key > current.key:
    if current.rightnode == None:
      current.rightnode = newNode
      newNode.parent = current
    else:
      insertNode(newNode, current.rightnode)

  if newNode.parent != None:
    return newNode.key
  else:
    return None


def nodeByKey(key, root):
  current = root
  if current == None:
    return None
  elif current.key == key:
    return current
  else:
    if nodeByKey(key, current.rightnode):
      return nodeByKey(key, current.rightnode)
    else:
      return nodeByKey(key, current.leftnode)


def deleteKey(B, key):
  if B.root == None:
    return None
  else:
    key = delKey(B.root, key)
    reBalance(B)
  return key


def delKey(root, key):
  if root != None:
    if root.key == key:
      if root.rightnode == None and root.leftnode == None:
        parent = root.parent
        if parent.key > key:
          parent.leftnode = None
        else:
          parent.rightnode = None
        return key

      elif root.rightnode == None and root.leftnode != None:
        parent = root.parent
        root.leftnode.parent = parent
        if key < parent.key:
          parent.leftnode = root.leftnode
        else:
          parent.rightnode = root.leftnode
        return key

      elif root.rightnode != None and root.leftnode == None:
        parent = root.parent
        root.rightnode.parent = parent
```

```
      if key < parent.key:
        parent.leftnode = root.rightnode
      else:
        parent.rightnode = root.rightnode
      return key

    else:
      successornode = inorderSucessor(root)
      root.value = successornode.value
      root.key = successornode.key

      if root.leftnode == successornode:
        root.leftnode = None
      else:
        delKey(root.leftnode, successornode.key)
      return key

  elif root.key < key:
    return delKey(root.rightnode, key)
  else:
    return delKey(root.leftnode, key)


def inorderSucessor(N):
  if isLeaf(N):
    parentNode = N.parent
    while parentNode.key < N.key:
      parentNode = parentNode.parent
    return parentNode
  N = N.rightnode
  if N:
    return successorNode(N)
  else:
    return None


def successorNode(node):
  if node.leftnode == None:
    return node
  else:
    return successorNode(node.leftnode)


def isLeaf(node):
  if node.leftnode == None and node.rightnode == None:
    return True
  else:
    return False


def reBalance(AVLTree):
  calculateBalance(AVLTree)
  reBalanceTree(AVLTree, AVLTree.root)
```

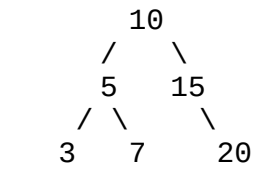
```
def reBalanceTree(Tree, root):
  if root.bf < 0:
    if root.rightnode.bf > 0:
      rotateRight(Tree, root.rightnode)
      rotateLeft(Tree, root)
    else:
      rotateLeft(Tree, root.rightnode)
  elif root.bf > 0:
    if root.leftnode.bf < 0:
      rotateLeft(Tree, root.leftnode)
      rotateRight(Tree, root)
    else:
      rotateRight(Tree, root.leftnode)


def preOrder(current, L):
  if current != None:
    add(L, current.bf)
    preOrder(current.leftnode, L)
    preOrder(current.rightnode, L)
  return L


def preOrden(current, L):
  if current != None:
    add(L, current.value)
    preOrden(current.leftnode, L)
    preOrden(current.rightnode, L)
  return L


def traversePreOrder(B):
  root = B.root
  preOrderList = LinkedList()
  preOrderList = preOrder(root, preOrderList)
  return preOrderList


def traversePreOrden(B):
  root = B.root
  preOrderList = LinkedList()
  preOrderList = preOrden(root, preOrderList)
  return preOrderList
```

LINK REPLIT: https://replit.com/@AlvaroBlanco/AVL-TREE-1

PARTE II

Ejercicio 6:

a. FALSO. Lo que se requiere es que para cada nodo del arbol la altura del subarbol derecho e izquierdo difieran en no mas de una unidad, esto no necesariamente implica que el penultimo nivel este completo:



En este ejemplo el nodo 15 pertenece al penultimo nivel el cual no esta completo pero el arbol esta balanceado

b. VERDADERO. Esto implica que las alturas de los subarboles de cada nodo del arbol son iguales en altura, ya que el balance factor es 0, lo cual nos indica que necesariamente el arbol debe estar completo para que cada nodo tenga iguales a sus respectivos subarboles.

c.VERDADERO. Si al actualizar el factor de balance del padre del nodo insertado no se produce un desequilibrio, esto significa que la diferencia de alturas entre el subárbol izquierdo y derecho del padre sigue siendo menor o igual a 1, lo que garantiza que el árbol sigue siendo AVL

d.VERDADERO. un arbol AVL puede tener una altura minima de 0, en cuyo caso consta unicamente de un nodo con factor de balance 0. Si un arbol AVL tiene una altura mayor que 0, entonces su hoja más profunda debe tener una altura minima de 0 o 1 (debido a la propiedad de equilibrio AVL), lo que implica que su padre tiene un factor de balance de -1, 0 o 1. Esto a su vez implica que hay al menos un nodo en el arbol con factor de balance 0.

Ejercicio 7:

Debido a que x sera una key que estara entre las key del arbol A (menores) y las key del arbol B (mayores). Deberia buscar el mas grande de las key en A (20) . Luego buscar el menor de las keys de B que son mayores a la key x (50). Ambas busquedas deberian ser en log n y log m respectivamente por ejemplo busqueda binaria. Luego deberia unir como subarbol izquierdo de la primer key (la key mas grande de A) y unir el subarbol derecho de la segunda key (key mas chica del arbol B mayor que x) con x como raiz.
La raiz del nuevo arbol sera x, con subarbol izquierdo a A y subarbol derecho a B.

Ejercicio 8:

Caso base: Para h=1, la altura del AVL es 1 y la única rama posible es la raíz. En este caso, la mínima longitud de una rama truncada es 0, que es igual a h/2 tomado la parte entera por abajo.

Hipótesis inductiva: Supongamos que para un AVL de altura h, la mínima longitud de una rama truncada es h/2 (tomando la parte entera por abajo).

Paso inductivo: Consideremos un AVL de altura h+1. Podemos tener dos casos:
1- El hijo derecho del nodo raíz está vacío. En este caso, la rama truncada de longitud mínima es la que sigue por el hijo izquierdo de la raíz. Como el subárbol izquierdo tiene altura h, por hipótesis inductiva la mínima longitud de una rama truncada es h/2 (tomando la parte entera por abajo). Sumando 1 a la longitud de esta rama, obtenemos una longitud mínima de (h/2)+1 para la rama truncada del AVL completo de altura h+1.
2- El hijo derecho del nodo raíz no está vacío. En este caso, la rama truncada de longitud mínima es la que sigue por el hijo derecho de la raíz y luego por la rama truncada de longitud mínima del subárbol derecho. Como ambos subárboles tienen altura h, por hipótesis inductiva la mínima longitud de una rama truncada en cada subárbol es h/2 (tomando la parte entera por abajo). Sumando 1 a la longitud de la rama truncada del hijo derecho, obtenemos una longitud mínima de (h/2)+1 para la rama truncada del AVL completo de altura h+1.