Tomás Rando - 14004 - Algo 2

Punto 1)

```
#Ejercicio 1
"""
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
"""
def rotateLeft(Tree, avlnode):

#Caso si el nodo es la raíz de todo el árbol
  if Tree.root == avlnode:
    Tree.root = avlnode.rightnode
    Tree.root.parent = None
    if Tree.root.leftnode != None:
      avlnode.rightnode = Tree.root.leftnode
      Tree.root.leftnode.parent = avlnode
    else:
      avlnode.rightnode = None
    Tree.root.leftnode = avlnode
    avlnode.parent = Tree.root
    return Tree.root

  #Verifico si el nodo esta a la derecha o a la izquierda del padre
  if avlnode.parent.rightnode == avlnode:
    padreHijo = "right"
  else:
    padreHijo = "left"

  #Inserto la nueva raiz dependiendo de PadreHijo
    avlnode.rightnode.parent = avlnode.parent
  if padreHijo == "right":
    avlnode.parent.rightnode = avlnode.rightnode
  else:
    avlnode.parent.leftnode = avlnode.rightnode
  #Si tiene hijo izquierdo, pasa a ser el hijo derecho de la antigua raiz. Posteriormente, la antigua raiz se inserta a la izquierda de la nueva raíz.
  newRoot = avlnode.rightnode
  if newRoot.leftnode != None:
    avlnode.rightnode = newRoot.leftnode
    newRoot.leftnode.parent = avlnode
  else:
    avlnode.rightnode = None
  newRoot.leftnode = avlnode
  avlnode.parent = newRoot
  return newRoot
"""
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
"""

def rotateRight(Tree, avlnode):
#Caso si el nodo es la raíz de todo el árbol
  if Tree.root == avlnode:
    Tree.root = avlnode.leftnode
    Tree.root.parent = None
    if Tree.root.rightnode != None:
      avlnode.leftnode = Tree.root.rightnode
      Tree.root.rightnode.parent = avlnode
    else:
      avlnode.leftnode = None
    Tree.root.rightnode = avlnode
    avlnode.parent = Tree.root
    return Tree.root

  #Verifico si el nodo esta a la derecha o a la izquierda del padre
  if avlnode.parent.rightnode == avlnode:
    padreHijo = "right"
  else:
    padreHijo = "left"

  #Inserto la nueva raiz dependiendo de PadreHijo
    avlnode.leftnode.parent = avlnode.parent
  if padreHijo == "right":
    avlnode.parent.rightnode = avlnode.leftnode
  else:
    avlnode.parent.leftnode = avlnode.leftnode
  #Si tiene hijo derecho, pasa a ser el hijo izquierdo de la antigua raiz. Posteriormente, la antigua raiz se inserta a la derecha de la nueva raíz.
  newRoot = avlnode.leftnode
  if newRoot.rightnode != None:
    avlnode.leftnode = newRoot.rightnode
    newRoot.rightnode.parent = avlnode
  else:
    avlnode.leftnode = None
  newRoot.rightnode = avlnode
  avlnode.parent = newRoot
  return newRoot
```

Punto 2)

```
#Ejercicio2
"""
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

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

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

  AVLNode.bf = height(AVLNode.leftnode) - height(AVLNode.rightnode)

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

```
#Función para calcular la altura del árbol
def height(currentNode):
  if currentNode == None:
    return 0
  else:
    Lheight = height(currentNode.leftnode)
    Rheight = height(currentNode.rightnode)

    if Lheight > Rheight:
      return Lheight + 1
    else:
      return Rheight + 1
```

Punto 3)

```
#Se recorre el árbol verificando los bf
def recorrerBf(AVLNode):

  if (AVLNode == None):
    return None

  if (abs(AVLNode.bf) != 1) and (AVLNode.bf != 0):
    return AVLNode


  left = recorrerBf(AVLNode.leftnode)
  if left != None:
    return left
  right = recorrerBf(AVLNode.rightnode)
  if right != None:
    return right
  return None

def reBalanceR(AVLTree, AVLNode):
  if (AVLNode.bf < -1):
    if AVLNode.rightnode != None:
      if AVLNode.rightnode.bf > 0:
        rotateRight(AVLTree, AVLNode.rightnode)
    rotateLeft(AVLTree, AVLNode)
  else:
    if (AVLNode.bf > 1):
      if AVLNode.leftnode != None:
        if AVLNode.leftnode.bf < 0:
          rotateLeft(AVLTree, AVLNode.leftnode)
      rotateRight(AVLTree, AVLNode)
  return AVLTree

def reBalance(AVLTree):
  AVLTree = calculateBalance(AVLTree)
  bfVerification = recorrerBf(AVLTree.root)
  if bfVerification != None:
    reBalanceR(AVLTree, bfVerification)
    reBalance(AVLTree)
  return AVLTree
```

Punto 4)

```
def updateBfAndFix(AVLTree, AVLNode):
  AVLNode.bf = height(AVLNode.leftnode) - height(AVLNode.rightnode)

  if (AVLNode.bf < -1):
    if AVLNode.rightnode != None:
      if AVLNode.rightnode.bf > 0:
        rotateRight(AVLTree, AVLNode.rightnode)
    rotateLeft(AVLTree, AVLNode)
  else:
    if (AVLNode.bf > 1):
      if AVLNode.leftnode != None:
        if AVLNode.leftnode.bf < 0:
          rotateLeft(AVLTree, AVLNode.leftnode)
      rotateRight(AVLTree, AVLNode)

  if AVLNode == AVLTree.root:
    return AVLTree
  else:
    updateBfAndFix(AVLTree, AVLNode.parent)
  return AVLTree

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
```

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

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

    if key != None:
      updateBfAndFix(AVLTree, newNode)
    return key
```

Punto 5)

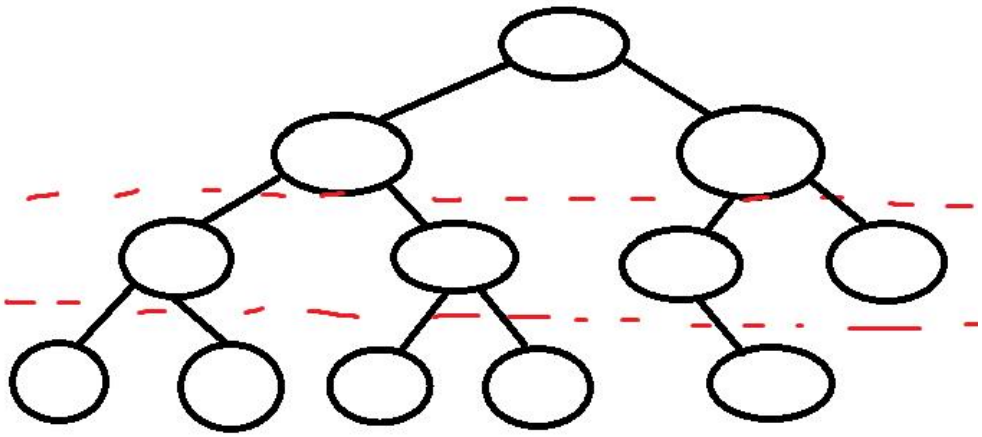
```
#Searchforelement y searchforminor buscan nodos utilizando recursividad que luego seran utilizados en la función delete
def searchforelement(currentNode, element):

  if currentNode == None:
    return None

  if currentNode.value == element:
    return currentNode

  left = (searchforelement(currentNode.leftnode, element))
  if left != None:
    return left

  right = (searchforelement(currentNode.rightnode, element))
  if right != None:
    return right

def searchforminor(currentNode):
  if currentNode.leftnode == None:
    if currentNode.rightnode != None:
      if currentNode.parent.leftnode == currentNode:
        currentNode.parent.leftnode = currentNode.rightnode
      else:
        currentNode.parent.leftnode = currentNode.rightnode
    else:
      if currentNode.parent.leftnode == currentNode:
        currentNode.parent.leftnode = None
      else:
        currentNode.parent.rightnode = None
    return currentNode
  else:
    return searchforminor(currentNode.leftnode)
```

```
def delete(AVLTree, element):
  node = searchforelement(AVLTree.root, element)
  if node != None:

    if (node.rightnode != None) and (node.leftnode != None):
      minornode= searchforminor(node.rightnode)
      if AVLTree.root == node:

        minornode.parent = None
        minornode.leftnode = AVLTree.root.leftnode
        minornode.rightnode = AVLTree.root.rightnode
        if minornode.rightnode != None:
          minornode.rightnode.parent = minornode
        if minornode.leftnode != None:
          minornode.leftnode.parent = minornode
        AVLTree.root = minornode

      else:
        if node.parent.rightnode == node:
          node.parent.rightnode = minornode
        if node.parent.leftnode == node:
          node.parent.leftnode = minornode
        minornode.parent = node.parent
        minornode.leftnode = node.leftnode
        minornode.rightnode = node.rightnode
        if node.leftnode != None:
          node.leftnode.parent = minornode
        if node.rightnode != None:
          node.rightnode.parent = minornode
```

```
    elif (node.rightnode == None) and (node.leftnode == None):

      if AVLTree.root == node:
        AVLTree.root = None
      else:

        if node.parent.leftnode == node:
          node.parent.leftnode = None
        else:
          node.parent.rightnode = None

    elif (node.rightnode == None) or (node.leftnode == None):
      if AVLTree.root == node:
        if AVLTree.root.rightnode != None:
          AVLTree.root = AVLTree.root.rightnode
          AVLTree.root.parent = None
        else:
          AVLTree.root = AVLTree.root.leftnode
          AVLTree.root.parent = None
      else:

        if node.parent.rightnode == node:
          if node.rightnode != None:
            node.parent.rightnode = node.rightnode
            node.parent.rightnode.parent = node.parent
          else:
            node.parent.rightnode = node.leftnode
            node.parent.rightnode.parent = node.parent
        else:
          if node.rightnode != None:
            node.parent.leftnode = node.rightnode
            node.parent.leftnode.parent = node.parent
          else:
            node.parent.leftnode = node.leftnode
            node.parent.leftnode.parent = node.parent
      updateBfAndFix(AVLTree, node.parent)
      return node.key
    else:
      return None
```

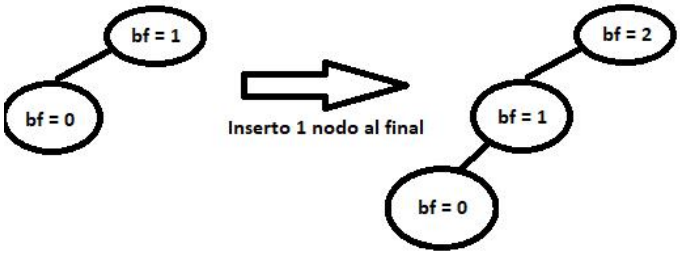
Punto 6)

a)



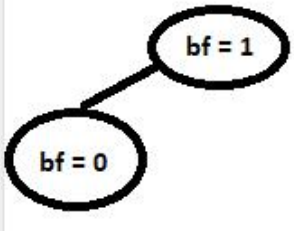
Este gráfico representa un AVL puesto que todos los bf de todos los nodos son correctos, es decir que el árbol se encuentra balanceado. Sin embargo, el penúltimo nivel no está completo pues no todos los nodos tienen dos hijos. Por ello, es falso

b)
Suponemos que es falsa. Entonces suponemos que existe un AVL que tiene todos los nodos con un bf = 0 y no es completo. Si no es completo, significa que hay un nodo que tiene solamente un hijo. Cuando eso sucede, significa que el balance factor de ese nodo es 1 o -1. Como contradice la hipótesis, es falso.
c)



Falso, en este ejemplo, se inserta un nodo al final del AVLTree, el bf del nodo del padre cambia a 1 (No desbalanceado), sin embargo, si hay que hacer rotaciones más arriba puesto que el bf de la raíz cambia a 2 (Está desbalanceada)

d) SIN HOJAS, es falso. Ya que por ejemplo se puede tener el siguiente ejemplo (Ver imagen) en el que el bf de la raíz es 1, y, al no contar las hojas, no existe ningún nodo con bf = 0.



Punto 7)
Para crear el algoritmo, primero se calculan las alturas de ambos árboles (A y B). Esto se puede hacer con una complejidad de (Log n)/(Log m) gracias al balanceFactor de los AVLTrees. Al tener esto, podemos contemplar 3 situaciones diferentes dependiendo de la altura calculada.

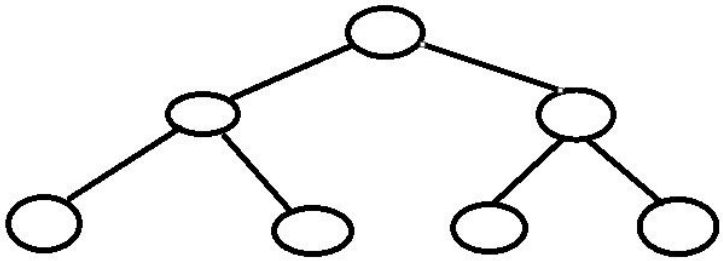
El más fácil es el caso 1, ambos árboles tienen la misma altura (o difieren en 1). En esta situación se puede usar x como la raíz del nuevo árbol, ya que los hijos derechos serán los del árbol B (Todos más grandes que este) y los izquierdos los del árbol A (Todos más chicos que x). Entonces quedará el árbol C colocando al nodo x como la raíz, a la raíz de B como el hijo derecho de x (la nueva raíz) y a la raíz de A como el hijo izquierdo de x.

En la situación 2, si la altura de B es mayor que la de A, se busca la altura de A y se va al nivel correspondiente en B. Ejemplo, si la altura de A = 3, se va al nivel 3 en B (Por los nodos más chicos). En ese nivel se coloca x como la "raíz" y el subárbol de ese nivel pasa a ser el hijo derecho de x. Y, todo A pasa a ser el hijo izquierdo de x.

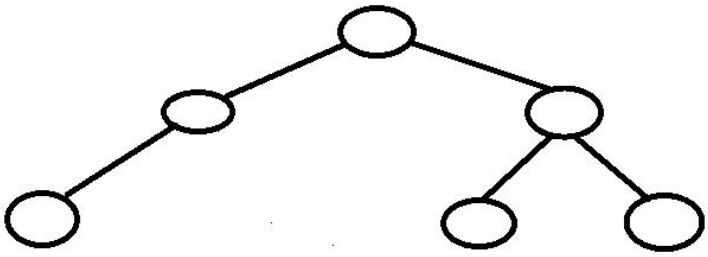
Por último, si la altura de A es mayor que la de B se hace lo mismo pero invertido, es decir, se busca la altura de B y nos dirigimos a el nivel correspondiente en A (Por los nodos más grandes). Se coloca x como raíz y ese subárbol de A pasa a estar conectado como nodo izquierdo de x. Además, se conecta a todo B con el lado derecho de la raíz.

Punto 8)

Suponemos que es falso. Entonces suponiendo que exista un árbol de altura h en el que la mínima ruta desde la raíz hasta el nodo con referencia none sea diferente de h/2, por ejemplo h. Entonces, si hacemos el esquema sería:



Sin embargo, podemos concluir que este enunciado es falso. Ya que puede existir una ruta diferente:



La cual es h/2. Como el enunciado negado es falso, significa que es verdadero que la mínima ruta desde la raíz hasta el nodo con referencia none en cualquier AVL es h/2.

Link al replit: https://replit.com/@TomasRando/TP-1-ALGO2
Todo el código también se encuentra en el repositorio de algoritmos2