Algoritmo y Estructura de Datos II

Alumno:Gonzalez Sanchez Gabriel
Legajo: 12007

Parte I

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

```
#Realiza una rotaciòn a derecha de un AVL
#O(1)
def rotateRight(Tree, avlnode):

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

  avlnode.parent = avlnode.leftnode
  if avlnode.leftnode.rightnode!=None:
    avlnode.leftnode.rightnode.parent = avlnode
  avlnode.leftnode = avlnode.leftnode.rightnode
  avlnode.parent.rightnode = avlnode
```

```
#Realiza una rotaciòn a izquierda de un AVL
#O(1)
def rotateLeft(Tree, avlnode):
  avlnode.rightnode.parent = avlnode.parent
  if avlnode.parent!=None:
    if avlnode.parent.rightnode == avlnode:
      avlnode.parent.rightnode = avlnode.rightnode
    else:
      avlnode.parent.leftnode = avlnode.rightnode
  else:
    Tree.root = avlnode.rightnode

  avlnode.parent = avlnode.rightnode
  if avlnode.rightnode.leftnode!=None:
    avlnode.rightnode.leftnode.parent = avlnode
  avlnode.rightnode = avlnode.rightnode.leftnode
  avlnode.parent.leftnode = avlnode
```

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.

```
#Ejercicio Nro 2
#Calcula el balance factor de cada nodo en un arbol
#Recorre el arbol en amplitud
#Devuelve el arbol con los valores de balance calculados
def calculateBalance(AVLTree):
  if AVLTree.root == None:
    return None
  else:
    L = LinkedList()
    insert(L, AVLTree.root, 0)
    calculateBalanceR(L, L.head, 1)
    printL(L)
    return AVLTree


def calculateBalanceR(L, currentNode, position):
  currentNode.value.bf = search_h(currentNode.value.leftnode) - search_h(
    currentNode.value.rightnode)
  if currentNode.value.leftnode != None:
    insert(L, currentNode.value.leftnode, position)
    position += 1
  if currentNode.value.rightnode != None:
    insert(L, currentNode.value.rightnode, position)
    position += 1
  if currentNode.nextNode != None:
    calculateBalanceR(L, currentNode.nextNode, position)
  currentNode.value = f'|Node: {currentNode.value.value} bf:
{currentNode.value.bf}|'
```

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.

```
#Ejercicio Nro 3
#Recorre un arbol binario en post-orden
#Reordena hacia arriba en caso de encontrar un nodo desbalanceado
def reBalance(AVLTree):
  if AVLTree.root == None:
    return None
  else:
    reBalanceR(AVLTree,AVLTree.root)
    return AVLTree


def reBalanceR(B,currentNode):
  if currentNode.leftnode != None:
    reBalanceR(B,currentNode.leftnode)
  if currentNode.rightnode != None:
    reBalanceR(B,currentNode.rightnode)
  if currentNode.bf!= 1 and currentNode.bf!=0 and currentNode.bf!=-1:
    sortIt(B,currentNode)
    calculateBalance(B)
```

```
#Aplica las rotaciones necesesarias para balancear un subarbol
def sortIt(B,currentNode):
  if currentNode.bf<0:
    #Caso especial
    if currentNode.rightnode.bf==1:
      rotateRight(B,currentNode.rightnode)
      rotateLeft(B,currentNode)
    #Desbalanceo hacia la derecha
    else:
      rotateLeft(B,currentNode)
  elif currentNode.bf>0:
    #Caso especial
    if currentNode.leftnode.bf==-1:
      rotateLeft(B,currentNode.leftnode)
      rotateRight(B,currentNode)
    #Desbalanceo hacia izquierda
    else:
      rotateRight(B,currentNode)
```

Ejercicio 4

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

```
#Ejercicio 4
#Inserta un nuevo nodo en un arbol AVL
#Busca la posicion donde insertar un nodo
#Asegura el balanceo en insercion
def insertBalanced(B, element, key):
  Node = AVLNode()
  Node.key = key
  Node.value = element
  Node.bf = 0
  if B.root == None:
    B.root = Node
  else:
    insertBalancedR(B,Node, B.root)

def insertBalancedR(B,newNode, currentNode):
  if newNode.key > currentNode.key:
    if currentNode.rightnode == None:
      currentNode.rightnode = newNode
      newNode.parent = currentNode
      update_bf(B,newNode,'insert')
      sortUp(B,newNode)
      return newNode.key
    else:
      return insertBalancedR(B,newNode, currentNode.rightnode)
  elif newNode.key < currentNode.key:
    if currentNode.leftnode == None:
      currentNode.leftnode = newNode
      newNode.parent = currentNode
      update_bf(B,newNode,'insert')
      sortUp(B,newNode)
      return newNode.key
    else:
      return insertBalancedR(B,newNode, currentNode.leftnode)
  else:
    return None
```

```
#Ordena cada nodo desde el actual hacia la raiz
def sortUp(Tree,currentNode):
  if currentNode!=None:
    if currentNode.bf!=0 and currentNode.bf!=1 and currentNode.bf!=-1:
      sortIt(Tree,currentNode)
    else:
      sortUp(Tree,currentNode.parent)

#Ordena el balance factor desde el nodo actual hacia la raiz
#Suma balance factor en caso de insercion y resta en caso de delete
def update_bf(Tree,currentNode,key):
  if currentNode!=None:
    if currentNode.parent!= None and key=='insert':
      if currentNode.parent.rightnode == currentNode:
        currentNode.parent.bf = currentNode.parent.bf-1
      elif currentNode.parent.leftnode == currentNode:
        currentNode.parent.bf = currentNode.parent.bf+1
      update_bf(Tree,currentNode.parent,'insert')

    elif currentNode.parent!=None and key=='delete':
      if currentNode.parent.leftnode == currentNode:
        currentNode.parent.bf = currentNode.parent.bf+1
      elif currentNode.parent.leftnode == currentNode:
        currentNode.parent.bf = currentNode.parent.bf-1
      update_bf(Tree,currentNode.parent,'delete')
```

Ejercicio 5

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

```
#Ejercicio 5
#Elimina la primera instancia de un elemento
#Asegura el balanceo del arbol
def deleteBalanced(B, element):
  if B.root == None:
    return None
  else:
    return deleteBalancedR(B, B.root, element)


def deleteBalancedR(B, currentNode, element):
  if currentNode.value == element:
    return delete_nodeBalanced(B, currentNode)
  else:
    if currentNode.leftnode != None:
      Left = deleteBalancedR(B, currentNode.leftnode, element)
      if Left != None:
        return Left
    if currentNode.rightnode != None:
      Right = deleteBalancedR(B, currentNode.rightnode, element)
      if Right != None:
        return Right




def delete_nodeBalanced(B, currentNode):
  #Prime caso: nodo sin hijos
```

```
if currentNode.leftnode == None and currentNode.rightnode == None:
  update_bf(B,currentNode,'delete')
  parent = currentNode.parent
  if currentNode.parent == None:
    B.root = None
  elif currentNode.parent.leftnode == currentNode:
    currentNode.parent.leftnode = None
  else:
    currentNode.parent.rightnode = None
  sortUp(B,parent)
#Segundo caso: El nodo tiene solamente un hijo
#Caso que el hijo este a la izquierda
elif currentNode.leftnode != None and currentNode.rightnode == None:
  update_bf(B,currentNode,'delete')
  parent = currentNode.parent
  if currentNode.parent == None:
    B.root = currentNode.leftnode
    B.root.parent = None
  elif currentNode.parent.leftnode == currentNode:
    currentNode.leftnode.parent = currentNode.parent
    currentNode.parent.leftnode = currentNode.leftnode
  else:
    currentNode.leftnode.parent = currentNode.parent
    currentNode.parent.rightnode = currentNode.leftnode
  sortUp(B,parent)
#Caso que el hijo este a la derecha
elif currentNode.leftnode == None and currentNode.rightnode != None:
  update_bf(B,currentNode,'delete')
  parent = currentNode.parent
  if currentNode.parent == None:
    B.root = currentNode.rightnode
    B.root.parent = None
  elif currentNode.parent.leftnode == currentNode:
    currentNode.rightnode.parent = currentNode.parent
    currentNode.parent.leftnode = currentNode.rightnode
  else:
    currentNode.rightnode.parent = currentNode.parent
    currentNode.parent.rightnode = currentNode.rightnode
  sortUp(B,parent)
#Caso que el nodo tenga dos hijos
else:
  #Busca el mayor de los menores
  findNode = currentNode.leftnode
  while findNode.rightnode != None:
```

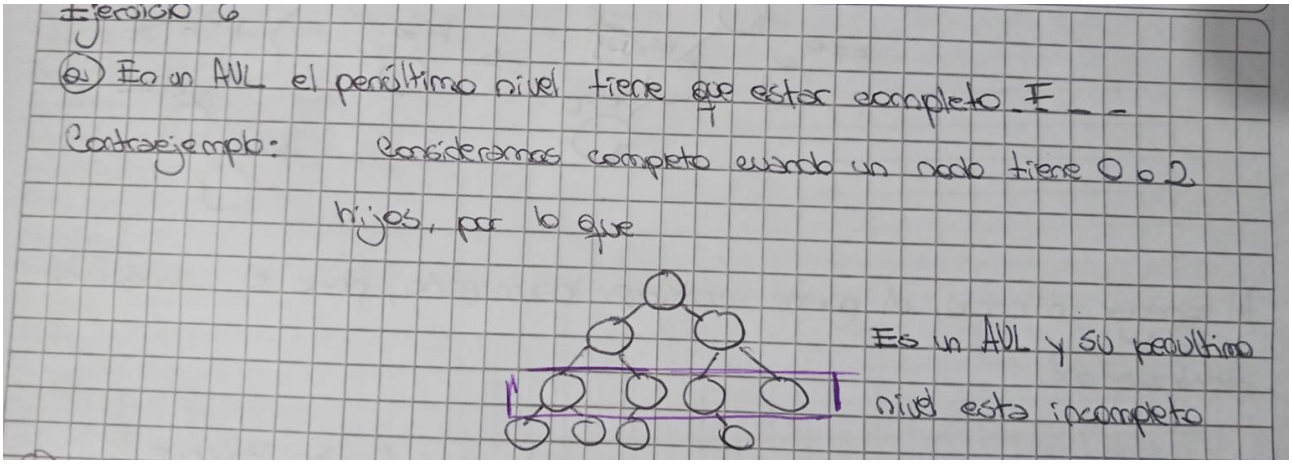
```
      findNode = findNode.rightnode
    currentNode.value = findNode.value
    return delete_nodeBalanced(B, findNode)
  return currentNode.key
```

Parte 2

Ejercicio 6

1. Responder V o F y justificar su respuesta:
    1. _F__ En un AVL el penúltimo nivel tiene que estar completo
    2. _V__ Un AVL donde todos los nodos tengan factor de balance 0 es completo
    3. _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.

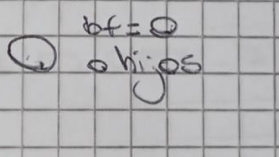
___ En todo *AVL* existe al menos un nodo con factor de balance 0.

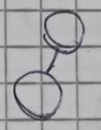


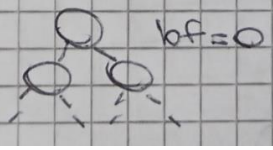
b) Un AVL donde todos los nodos tengan un factor de balance 0 es completo. - V - -

Supongamos el caso contrario. Si un AVL donde todos sus nodos son bf = 0 entonces es incompleto.

Es falso ya que sus nodos poseen 0 nodos o cada rama de ese nodo poseen un balance por que tendrá 0 o 2 hijos y el árbol quedara completo

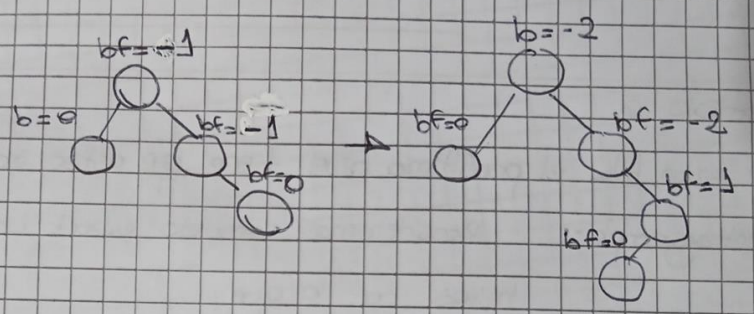






c) En la inserción en un AVL, si al actualizarse el factor de balance al padre del nodo insertado éste no se desbalanceó, entonces no hay que seguir verificando hacia arriba porque no hay cambio en los factores de balance.
- F - -

Contraejemplo:

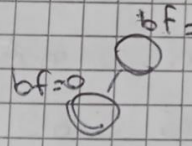


Al agregar un nodo, el padre continua balanceado, pero el "abuelo" se desbalanceo.

d) En todo AVL existe al menos un nodo con factor balance 0.

— V — —

Todas las hojas poseen factor de balance 0. Si ignoramos el caso de las hojas un caso posible es el de un árbol con dos nodos:



De esta forma demostramos que no en todo AVL hay un nodo con bf=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 $a \in A$ 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*.