Licenciatura Ciencias de la computación - Uncuyo
Augusto Robles 11737

Trabajo Práctico Árboles balanceados AVL

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):
    # El nuevo nodo raiz es el hijo derecho de la antigua raiz
    newRootNode = avlnode.rightnode
    if avlnode.parent != None:
        avlnode.parent.rightnode = newRootNode
    avlnode.rightnode = None
    # Si el nodo raiz anterior era la raiz del arbol, asignar su hijo derecho como nueva raiz del arbol
    if avlnode.parent == None:
        Tree.root = newRootNode
        newRootNode.parent = None
    else:
        # Sino asigno como padre del nuevo nodo raiz al padre del antiguo nodo raiz
        newRootNode.parent = avlnode.parent
    # Si el hijo derecho de la antigua raiz tenia un hijo izquierdo, este pasa a ser hijo derecho de la antigua raiz
    if newRootNode.leftnode != None:
        avlnode.rightnode = newRootNode.leftnode
        newRootNode.leftnode.parent = avlnode
    # El antiguo nodo raiz pasa a ser el hijo izquierdo del nuevo nodo raiz
    newRootNode.leftnode = avlnode
    avlnode.parent = newRootNode
    # Retorno la nueva raiz del arbol
    return newRootNode
```

```python
def rotateRight(Tree,avlnode):
    # El nuevo nodo raiz es el hijo izquierdo de la antigua raiz
    newRootNode = avlnode.leftnode
    if avlnode.parent != None:
        avlnode.parent.leftnode = newRootNode
    avlnode.leftnode = None
    # Si el nodo raiz anterior era la raiz del arbol, asignar su hijo derecho como nueva raiz del arbol
    if avlnode.parent == None:
        Tree.root = newRootNode
        newRootNode.parent = None
    else:
        # Sino asigno como padre del nuevo nodo raiz al padre del antiguo nodo raiz
        newRootNode.parent = avlnode.parent
    # Si el hijo izquierdo de la antigua raiz tenia un hijo derecho, este pasa a ser hijo izquierdo de la antigua raiz
    if newRootNode.rightnode != None:
        avlnode.leftnode = newRootNode.rightnode
        newRootNode.rightnode.parent = avlnode
    # El antiguo nodo raiz pasa a ser el hijo derecho del nuevo nodo raiz
    newRootNode.rightnode = avlnode
    avlnode.parent = newRootNode
    # Retorno la nueva raiz del arbol
    return newRootNode
```

Licenciatura Ciencias de la computación - Uncuyo
Augusto Robles 11737

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 findHeight(node):
    # Caso base
    if node == None:
        return -1
    # Casos recursivos
    leftHeight = findHeight(node.leftnode)
    rightHeight = findHeight(node.rightnode)
    # Comparo las alturas y retorno recursivamente la mayor entre la izq y la der
    if leftHeight >= rightHeight:
        return 1 + leftHeight
    return 1 + rightHeight

def calcBalanceR(avlnode):
    # Caso base
    if avlnode == None:
        return 0
    # Casos recursivos
    leftHeight = calcBalanceR(avlnode.leftnode)
    rightHeight = calcBalanceR(avlnode.rightnode)
    # Calculo y actualizo el balance factor del nodo
    avlnode.bf = leftHeight - rightHeight
    # El retorno recursivo lleva implicito el valor de la altura del nodo mediante una funcion recursiva que calcula la altura
    return 1 + findHeight(avlnode)


def calculateBalance(AVLTree):
    if AVLTree.root == None:
        return
    # Llamado a funcion recursiva
    calcBalanceR(AVLTree.root)
    return AVLTree
```

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 reBalanceR(AVLTree, avlnode):
    # Caso base
    if avlnode == None:
        return
    # Detecto si el nodo tiene desbalance hacia la derecha
    if avlnode.bf < -1:
        # Se detecta si el hijo derecho tiene un hijo izquierdo
        if avlnode.rightnode.bf > 0:
            rotateRight(AVLTree, avlnode.rightnode)
            avlnode = rotateLeft(AVLTree, avlnode)
        else:
            avlnode = rotateLeft(AVLTree, avlnode)
        calculateBalance(AVLTree)
    # Detecto si el nodo tiene desbalance hacia la izquierda
    elif avlnode.bf > 1:
        # Se detecta si el hijo izquierdo tiene un hijo derecho
        if avlnode.leftnode.bf < 0:
            rotateLeft(AVLTree, avlnode.leftnode)
            avlnode = rotateRight(AVLTree, avlnode)
        else:
            avlnode = rotateRight(AVLTree, avlnode)
        calculateBalance(AVLTree)
    # Llamadas recursivas
    reBalanceR(AVLTree, avlnode.leftnode)
    reBalanceR(AVLTree, avlnode.rightnode)

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

Licenciatura Ciencias de la computación - Uncuyo
Augusto Robles 11737

Ejercicio 4:

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

```python
def insert(AVLTree,element,key):
    current = AVLTree.root
    # Creo un nuevo nodo con la key y value ingresada por el usuario
    newNode = AVLNode()
    newNode.key = key
    newNode.value = element
    if current == None:
        AVLTree.root = newNode
        AVLTree.root.bf = 0
        return key
    recursiveInsert(newNode,AVLTree.root)
    # Una vez insertado el nodo, calculo el bf de cada nodo y balanceo desde el nodo insertado hasta la raiz del arbol
    updateBfAndBalance(AVLTree, newNode)


def recursiveInsert(newNode, treeNode):
    if newNode.key > treeNode.key:
        if treeNode.rightnode == None:
            treeNode.rightnode = newNode
            newNode.parent = treeNode
        else:
            recursiveInsert(newNode, treeNode.rightnode)
    else:
        if treeNode.leftnode == None:
            treeNode.leftnode = newNode
            newNode.parent = treeNode
        else:
            recursiveInsert(newNode, treeNode.leftnode)
```

```python
def updateBfAndBalance(tree, node):
    if node == None:
        return
    calcBalanceR(node)
    if abs(node.bf) > 1:
        reBalanceR(tree, node)
        return
    # Llamado recursivo
    updateBfAndBalance(tree, node.parent)
```

Ejercicio 5:

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

```python
def delete(B,element):
    nodeToDelete = searchR(B.root,element)
    flag = 0
    if nodeToDelete == B.root:
        flag = 1
    if nodeToDelete == None:
        return

    if nodeToDelete.leftnode == None:
        transplant(B,nodeToDelete,nodeToDelete.rightnode)

    elif nodeToDelete.rightnode == None:
        transplant(B,nodeToDelete,nodeToDelete.leftnode)

    else:
        y = treeMinimum(nodeToDelete.rightnode)
        if y.parent != nodeToDelete:
            transplant(B,y,y.rightnode)
            y.rightnode = nodeToDelete.rightnode
            y.rightnode.parent = y
        transplant(B,nodeToDelete,y)
        y.leftnode = nodeToDelete.leftnode
        y.leftnode.parent = y
    if flag == 1:
        reBalance(B)
    else:
        updateBfAndBalance(B, nodeToDelete.parent)
    return nodeToDelete.key
```

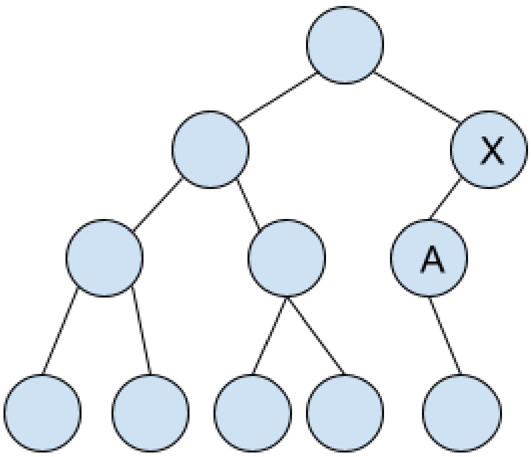
```python
def searchR(node, element):
    if node == None:
        return
    if node.value == element:
        return node
    right = searchR(node.rightnode, element)
    if right != None:
        return right
    left = searchR(node.leftnode, element)
    if left != None:
        return left

def treeMinimum(node):
    while node.leftnode != None:
        node = node.leftnode
    return node

def treeMaximum(node):
    while node.rightnode != None:
        node = node.rightnode
    return node

def transplant(B,u,v):
    if u.parent == None:
        B.root = v
    elif u == u.parent.leftnode:
        u.parent.leftnode = v
    else:
        u.parent.rightnode = v
    if v != None:
        v.parent = u.parent
```

Parte 2

Ejercicio 6:

1. Responder V o F y justificar su respuesta:
   a. ___ En un AVL el penúltimo nivel tiene que estar completo
   b. ___ Un AVL donde todos los nodos tengan factor de balance 0 es completo
   c. ___ 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.
   d. ___ En todo *AVL* existe al menos un nodo con factor de balance 0.

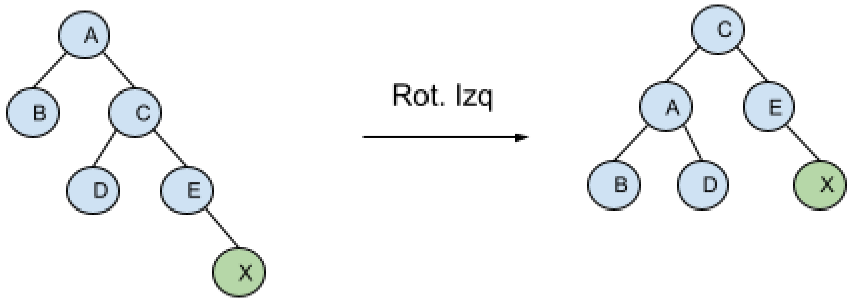
a) **Verdadero.**
Demostración: Supongamos un árbol AVL que tiene su penúltimo nivel incompleto.



Existe un nodo X, tal que es el antepenúltimo nodo. Este tiene un hijo A que no es hoja, por tanto A tiene al menos un hijo. Luego el balance factor del nodo X es 2 o -2 y el árbol no es AVL (contradicción). Luego la sentencia a) es verdadera.

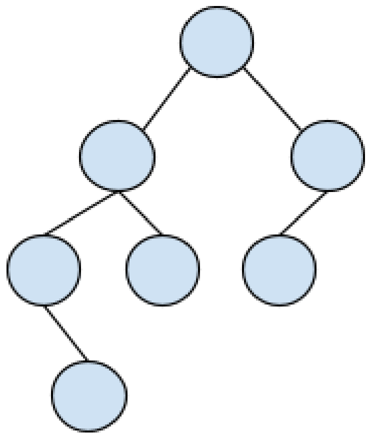
b) **Verdadero**
Demostración: Supongamos un AVL donde cada nodo tiene un balance factor = 0 y no es completo. Existe entonces un nodo que tiene solo un hijo y el balance factor != 0 para ese nodo. Por contradicción b) es verdadera.

c) **Falso**
Demostración:



Al insertar el nodo X en un árbol balanceado, se actualiza el bf al padre del nodo insertado y este no queda desbalanceado, bf(E) = -1. Pero al seguir verificando hacia arriba vemos que el bf(A) = -2, luego existió cambio de bf y el árbol quedó desbalanceado, siendo necesario rebalancear con una rotación hacia la izquierda.

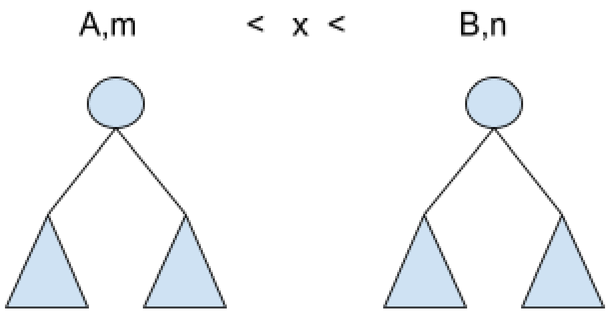
d) **Falso**
Demostración:



El árbol es un AVL y para todo nodo que no sea hoja (bf=0) no existe ningún nodo con bf = 0.

Ejercicio 7:

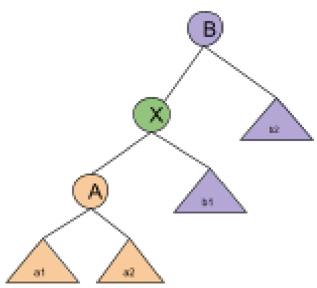
Sean *A* y *B* dos *AVL* de *m* y *n* nodos respectivamente y sea *x* un key cualquiera de forma tal que para todo key $a \in A$ y para todo key $b \in B$ se cumple que $a < x < b$. Plantear un algoritmo $O(\log n + \log m)$ que devuelva un *AVL* que contenga los key de *A*, el key *x* y los key de *B*.



1) Calcular las alturas de los dos árboles A y B. Determinar cuál es el de mayor tamaño.
2) Si el de mayor tamaño es el árbol A, se insertará el nodo con la key x en el subárbol izquierdo del árbol B. Si el de mayor tamaño es el árbol B, se insertará el nodo con la key x en el subárbol derecho de A.

   Suponiendo el caso donde el árbol de mayor altura es el B:

3) Inserto el nodo con el key x en el árbol B, en el lugar del nodo de B cuya altura es la misma que tiene el árbol A, tomada desde las hojas de B hacia arriba.
4) Inserto el árbol A como hijo izquierdo de X.
5) Desconecto el subárbol B1 del lugar donde se insertó X e insertarlo como hijo derecho de X
6) X será raíz de un subárbol AVL porque los subárboles varían como máximo en una unidad de altura.
7) Solo se deberá rebalancear si es necesario desde el nodo X hacia arriba, lo que ahorraría gran cantidad de rebalanceos.



Cálculo altura A: log m
Cálculo altura B: log n
Inserto X en B: log n
Operaciones de desconectar y conectar subárboles: O(1)
Inserto Árbol A en X: log n
Luego => 3 log n + log m = **O(log n + log m)**

Licenciatura Ciencias de la computación - Uncuyo
Augusto Robles 11737

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.