Martins Ezequiel 14000

Trabajo Práctico Arboles AVL (U2)

https://replit.com/@EzeMarts/tp-arboles#avltree.py

Ejercicio 1

```
def rotateLeft(Tree, AVLnode):
  nodeA = AVLnode
  nodeB = nodeA.rightnode
  nodeA.rightnode = nodeB.leftnode
  if nodeB.leftnode != None:
    nodeB.leftnode.parent = nodeA
  nodeB.parent = nodeA.parent
  if nodeA.parent == None:
    Tree.root = nodeB
  elif nodeA == nodeA.parent.leftnode:
    nodeA.parent.leftnode = nodeB
  else:
    nodeA.parent.rightnode = nodeB
  nodeB.leftnode = nodeA
  nodeA.parent = nodeB
  return Tree.root
```

```
def rotateRight(Tree, AVLnode):
  nodeA = AVLnode
  nodeB = nodeA.leftnode
  nodeA.leftnode = nodeB.rightnode
  if nodeB.rightnode != None:
    nodeB.rightnode.parent = nodeA
  nodeB.parent = nodeA.parent
  if nodeA.parent == None:
    Tree.root = nodeB
  elif nodeA == nodeA.parent.rightnode:
    nodeA.parent.rightnode = nodeB
  else:
    nodeA.parent.leftnode = nodeB
  nodeB.rightnode = nodeA
  nodeA.parent = nodeB
  return Tree.root
```

Ejercicio 2

```
def calculateBalance(AVLTree):
  if AVLTree.root == None:
    return None
  calculateBalanceR(AVLTree.root)
  return AVLTree
def calculateBalanceR(Node):
  if Node == None:
    return
  Node.bf = height(Node.leftnode) - height(Node.rightnode)
  calculateBalanceR(Node.leftnode)
  calculateBalanceR(Node.rightnode)
def height(Node):
  if Node == None:
    return 0
  hleft = height(Node.leftnode)
  hright = height(Node.rightnode)
  h = max(hleft, hright) + 1
  return h
```

Ejercicio 3

```
def reBalance(AVLTree):
  if AVLTree.root == None:
    return None
  calculateBalance(AVLTree)
  while AVLTree.root.bf < -1 or AVLTree.root.bf > 1:
    reBalanceR(AVLTree, AVLTree.root)
  return
def reBalanceR(AVLTree, Node):
  if Node.bf >= -1 and Node.bf <= 1:
    holdNode = Node.parent
    if Node.parent == None:
      return
    if holdNode.bf < 0:
      if holdNode.rightnode.bf > 0:
        rotateRight(AVLTree, holdNode.rightnode)
        rotateLeft(AVLTree, holdNode)
      else:
        rotateLeft(AVLTree, holdNode)
    elif holdNode.bf > 0:
      if holdNode.leftnode.bf < 0:
        rotateLeft(AVLTree, holdNode.leftnode)
        rotateRight(AVLTree, holdNode)
      else:
        rotateRight(AVLTree, holdNode)
    calculateBalance(AVLTree)
    return
  else:
    if Node.bf > 1:
      reBalanceR(AVLTree, Node.leftnode)
    else:
      reBalanceR(AVLTree, Node.rightnode)
  return
```

Ejercicio 4

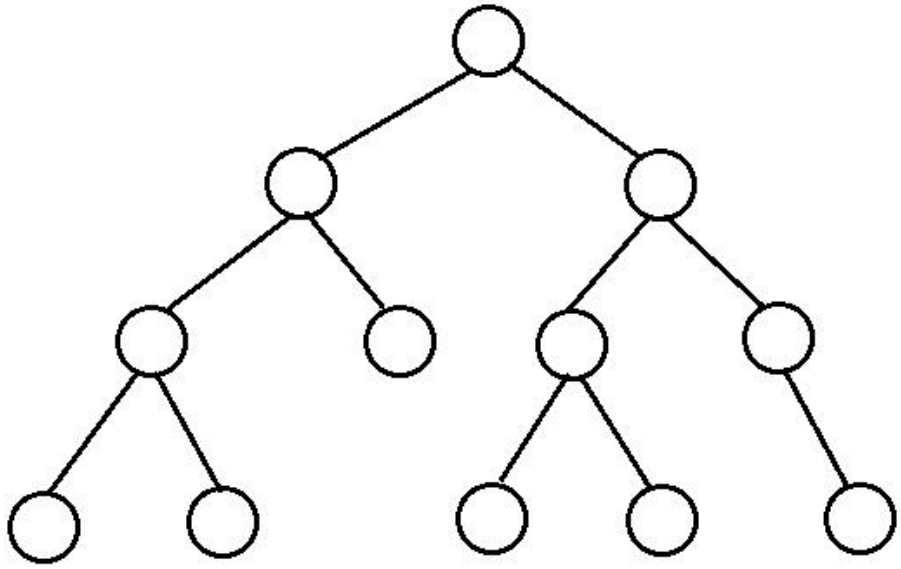
```
def insert(AVLTree, value, key):  # Inserta un nodo con value y key
  newNode = AVLNode()
  newNode.key = key
  newNode.value = value
  NodeA = None
  NodeB = AVLTree.root
  while NodeB != None:
    NodeA = NodeB
    if newNode.key < NodeB.key:
      NodeB = NodeB.leftnode
    else:
      NodeB = NodeB.rightnode
  newNode.parent = NodeA
  if NodeA == None:
    AVLTree.root = newNode
  elif newNode.key < NodeA.key:
    NodeA.leftnode = newNode
  else:
    NodeA.rightnode = newNode
  reBalance(AVLTree)
  return
```

Ejercicio 5

```
def delete(AVLTree, value):  # Elimina un nodo por su value y devuelve
  Node = AVLTree.root
  if Node == None:
    return None
  while Node.value != value:
    if Node.value > value:
      Node = Node.leftnode
    else:
      Node = Node.rightnode
    if Node == None:
      return None
  if Node.leftnode == None:
    rearrangeTree(AVLTree, Node, Node.rightnode)
  elif Node.rightnode == None:
    rearrangeTree(AVLTree, Node, Node.leftnode)
  else:
    newNode = minNode(Node.rightnode)
    if newNode.parent != Node:
      rearrangeTree(AVLTree, newNode, newNode.rightnode)
      newNode.rightnode = Node.rightnode
      newNode.rightnode.parent = newNode
    rearrangeTree(AVLTree, Node, newNode)
    newNode.leftnode = Node.leftnode
    newNode.leftnode.parent = newNode
  reBalance(AVLTree)
  return Node.key
def rearrangeTree(AVLTree, NodeA, NodeB):
  if NodeA.parent == None:
    AVLTree.root = NodeB
  elif NodeA == NodeA.parent.leftnode:
    NodeA.parent.leftnode = NodeB
  else:
    NodeA.parent.rightnode = NodeB
  if NodeB != None:
    NodeB.parent = NodeA.parent
  return
def minNode(Node):
  while Node.leftnode != None:
    Node = Node.leftnode
  return Node
def maxNode(Node):
  while Node.rightnode != None:
    Node = Node.rightnode
  return Node
```

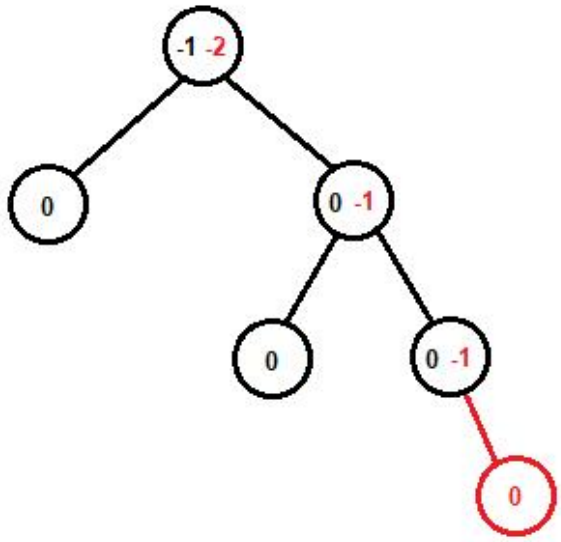
Ejercicio 6

a. **F** En un AVL no es necesario que el penúltimo nivel este completo, ya que su BF solo variara entre -1 y 1. En la imagen se puede apreciar un ejemplo de esto

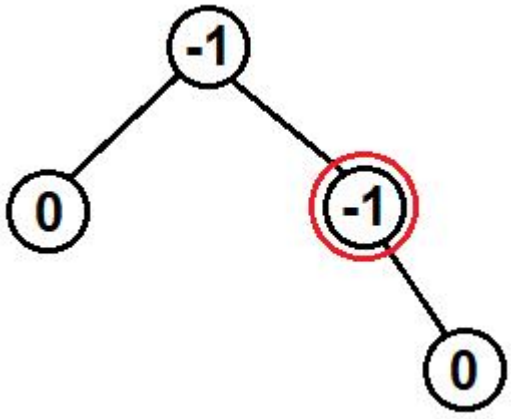


b. V Un AVL Completo es un árbol en el que todos los niveles tienen nodos de cero *[nodos hojas]* o dos hijos, es decir, que solo pueden tomar de valor 0 como BF. En el caso de que un nodo tenga un BF distinto de 0 significa que tiene un solo hijo y por lo tanto no es completo.

c. F Hay que seguir verificando el BF de los nodos superiores porque se puede producir un desbalance lejos de la inserción, en la imagen se ve como cerca de la inserción no hay desbalance pero cuando se sube al .root con parent se puede ver un desbalance.

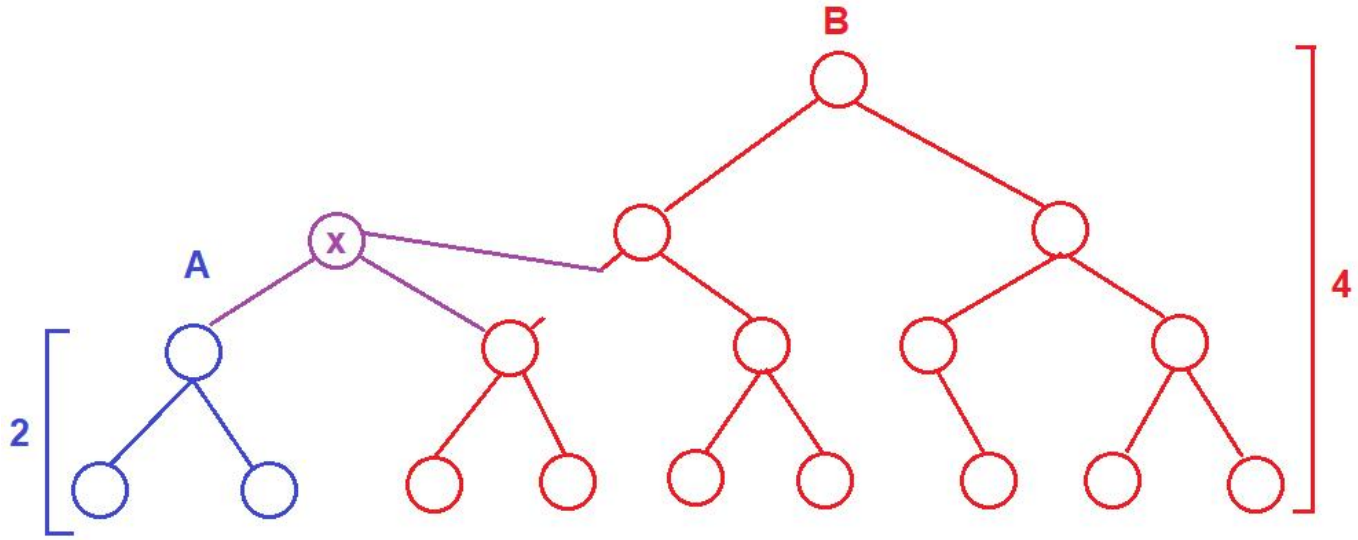


d. F *[Sin contar los nodos hoja y raíz]* Existen los casos donde, desde la raiz, pueden haber 1 nodo hacia un lado y 2 hacia el otro, quedando balanceado el árbol pero ese nodo "intermedio" con BF de valor 1 o -1.



Ejercicio 7

Para este algoritmo primero se debe de calcular la altura de ambos AVL, esta operación es de O(log n) por lo que no altera nuestro resultado. Luego de calcular la altura de ambos se compara cual de los dos es el de mayor altura, tras eso se hace una diferencia entre el mayor y el menor, esto para saber cuantos niveles hay que bajar para hacer el enlace. Luego de bajar los niveles necesarios, se hace un cambio de parent, para el nodo de menor altura su raíz se enlaza con el nodo de valor x, y en el árbol de mayor altura, desde el nodo donde estamos parados, hacemos cambios de parent e hijos.



Ejercicio 8

Vamos a tomar que esto se cumple para todos los BT AVL que tengan el penúltimo nivel lleno. Partiendo de un caso base donde el 1er nodo este solitario, siendo el número de aristas 0 y su altura 1, se cumple que la mínima longitud de la rama trunca trunc(1/2)=0, el siguiente caso donde hayan dos nodos (raíz e hijo) siendo sus aristas 1 y su altura 2, cumpliendo que la mínima longitud de la rama truncada trunc(2/2)=1, para el caso donde hayan tres nodos (raíz y dos hijos) se cumple lo mismo que con dos hijos, y ya a partir de este punto mientras más nodos se añadan y respetando que sea AVL y su penúltimo nodo lleno, siempre la mínima longitud de su rama truncada dará la altura/dos *(ya que estos arboles se pueden armar juntando los tres casos base)*. En la imagen se puede ver como se construyo un árbol juntando varias veces el 2do y 3er caso.