

<https://replit.com/@LuceroFederico/Ejercicios-Arboles-Balanceados#avltree.py>

Parte 1

Importante: Los ejercicios de esta primera parte tienen como objetivo codificar las diferentes funciones básicas necesarias para la implementar un árbol AVL.

A partir de estructuras definidas como :

```
class AVLTree:  
    root = None  
  
class AVLNode:  
    parent = None  
    leftnode = None  
    rightnode = None  
    key = None  
    value = None  
    bf = None
```

Copiar y adaptar todas las operaciones del **binarytree.py** (i.e insert(), delete(), search(),etc) al nuevo módulo **avltree.py**. Notar que estos luego deberán ser implementados para cumplir que la propiedad de un árbol 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

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 subárbol

Ejercicio 3

Implementar una función en el módulo `avltree.py` de acuerdo a las siguientes especificaciones:

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 4:

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

Ejercicio 5:

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

Parte 2

Ejercicio 6:

1. Responder V o F y justificar su respuesta:

- a. F En un AVL el penúltimo nivel tiene que estar completo
- b. V Un AVL donde todos los nodos tengan factor de balance 0 es completo
- c. 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
- d. F En todo AVL existe al menos un nodo con factor de balance 0.

Ejercicio 6

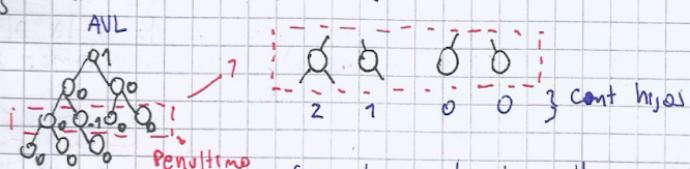
a. F

Suponiendo que Nivel completo significa que $2^h = \text{Cantidad de nodos}$ en el Nivel, $h = \text{Altura de Nivel}$

h	AVL	según h Cantidad Nodos	Nodos Cantidad
0		2^0	1
1		2^1	2
2		2^2	3
3		2^3	1

Penúltimo Nivel \rightarrow según lo supuesto, el penúltimo nivel está incompleto

Suponiendo que Nivel completo significa que cada nodo debe tener 2 hijos



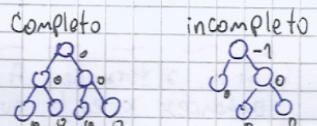
Penúltimo Nivel \rightarrow según lo supuesto el penúltimo nivel está incompleto

b. V

Árbol AVL Completo = Cada nodo tiene 2 hijos

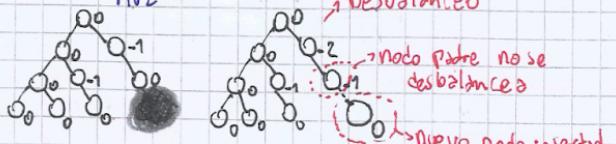
$$\begin{cases} \text{Si Cada nodo tiene } bf=0 \text{ entonces} \\ bf = h(\text{left subtree}) - h(\text{right subtree}) = 0 \end{cases}$$

$h(\text{left subtree}) = h(\text{right subtree})$, por lo tanto las alturas difieren en 0 unidades



c. F

AVL Desbalanceo

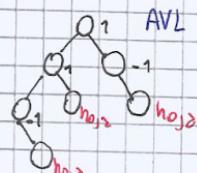


deberás subir hasta la raíz para verificar que el árbol no se desbalanceó

Si el bf difiere de $\{0, 1, -1\}$ este se considera desbalanceado

d. F

Sin considerar las hojas

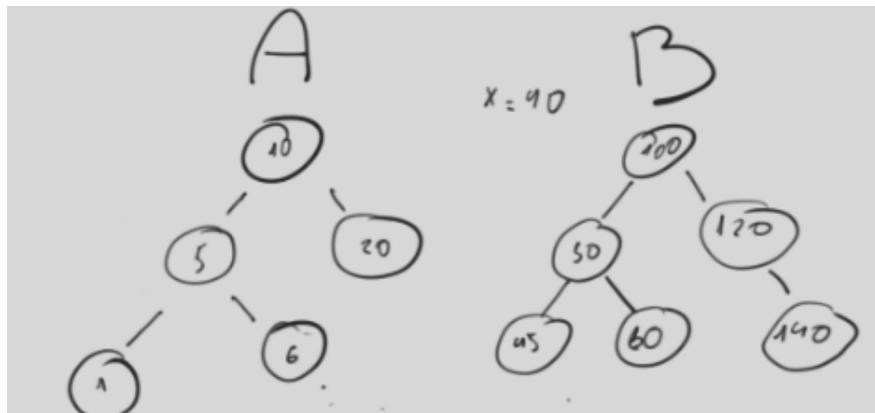


Si el nodo es una hoja entonces su bf = 0

En este caso el bf de todos los nodos, menos las hojas, valen 1 o -1

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 .



Ejercicio 7

$A \leftarrow X \leftarrow B$

Algoritmo de $O(\log n + \log m)$
 m nodos $\in A$
 n nodos $\in B$
 X = numero

$\text{log} = \text{log}$ seguida
 en otra

{ Buscamos la altura de A y la altura de B ($\log n + \log m$), luego insertamos X y reasignamos nodos }

{ Si la altura de A es menor que la altura de B se inserta como se ve en la imagen, B es la nueva raiz }

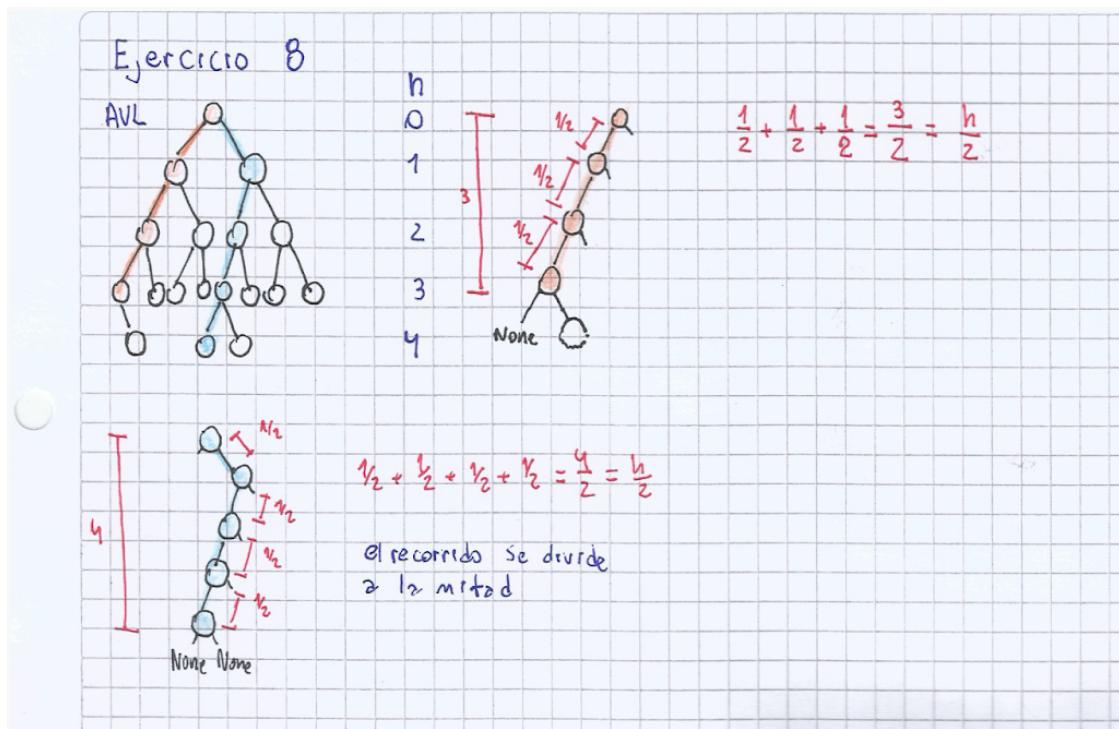
{ Si la altura de A = altura de B entonces X es la nueva raiz }

{ Si la altura de A es mayor que la altura de B se inserta como se ve en la imagen, A es la nueva raiz }

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.



Parte 3

Ejercicios Opcionales

- Si n es la cantidad de nodos en un árbol AVL, implemente la operación `height()` en el módulo `avltree.py` que determine su altura en $O(\log n)$. Justifique el por qué de dicho orden.

```
def calculate_height(avlnode):
    if avlnode == None:
        return 0
    avlnode.h = max(calculate_height(avlnode.leftnode),calculate_height(avlnode.rightnode)) + 1
    return avlnode.h
```

- Considere una modificación en el módulo `avltree.py` donde a cada nodo se le ha agregado el campo `count` que almacena el número de nodos que hay en el subárbol en el que él es raíz. Programe un algoritmo $O(\log n)$ que determine la cantidad de nodos en el árbol cuyo valor del key se encuentra en un intervalo $[a, b]$ dado como

parámetro. Explique brevemente por qué el algoritmo programado por usted tiene dicho orden.

A tener en cuenta:

- 1. Usen lápiz y papel primero**
- 2. No se puede utilizar otra Biblioteca mas alla de algo1.py y las bibliotecas desarrolladas durante Algoritmos y Estructuras de Datos I.**

Bibliografia:

[1] Guido Tagliavini Ponce, [Balanceo de arboles y arboles AVL](#) (Universidad de Buenos Aires)

[2] Brad Miller and David Ranum, Luther College, [Problem Solving with Algorithms and Data Structures using Python](#).