**Ejercicio de Arboles AVL (Balanceo)**

Este ejercicio es necesario seguirlo, muy atentamente, a través de los dibujos, y línea a línea los códigos fuentes de las distintas funciones. Planteo el caso de una inserción donde el desbalanceo queda en forma de L (por lo tanto tenemos que aplicar una doble rotación). Si hay dudas con este concepto, por favor me consultan.

Caso: Supongamos que crearemos un árbol ingresando los siguientes nodos en orden **4 2 3 1**

**Paso 1.**

Ingresamos el 4. El árbol quedará balanceado porque tiene orden de balanceo 0. Esto es: alto 0 para su subárbol izquierdo, y alto 0 para el derecho.



**Paso 2.**

Ingresamos el 2. Como es menor que el cuatro se inserta en el subárbol izquierdo, que por ahora es vacío. Por lo tanto el 2 quedará como raíz del subárbol izquierdo.

El nodo 4 queda con un grado de balanceo 1, porque resto el alto del subárbol izquierdo al alto del subárbol derecho. Como se admite hasta un grado de balanceo **|gb| <= 1**, al nodo 4 todavía se lo considera balanceado. El nodo 2 también queda balanceo ya que sus árboles izquierdo y derecho tienen el mismo alto: 0.



**Paso 3.**

Ingresamos el 3. El orden de inserción sigue por el subárbol izquierdo del 4 (ya que 3 es menor que 4), y se inserta como raíz del subárbol derecho de 2.



En este caso 3 queda balanceado, 2 queda balanceado, ya que tiene gb=-1. Pero ahora 4 queda desbalanceado, ya que su subárbol derecho tiene un alto de 2, el subárbol izquierdo tiene un alto de 0, y la resta dará un gb=2 para el nodo 4. En este momento se aplica la función de balanceo al nodo 4. Aplicaremos la función del siguiente algoritmo al nodo 4.

Se cumple este if más abajo en **insertar**.

|  |
| --- |
| if (balance > 1 && valor > Nodo->izq->valor)  {  Nodo->izq = rotaIzquierda(Nodo->izq);  return rotaDerecha(Nodo);  } |

Enconces llama primero a rotaIzquierda con el nodo 2 y luego a rotaDerecha. (doble rotación porque la rama quedo en forma de L).

|  |
| --- |
| **struct Nodo \*rotaIzquierda(struct Nodo \*x)** {     struct Nodo \*y = x->der;     struct Nodo \*T2 = y->izq;     y->izq = x;     x->der = T2;     x->alto = max(alto(x->izq), alto(x->der))+1;     y->alto = max(alto(y->izq), alto(y->der))+1;     return y; } |

    struct Nodo \*y = x->der;  
    struct Nodo \*T2 = y->izq;



y->izq = x;

x->der = T2;

queda



La función retorna **y,** y este nodo se asigna en 4->izq, y queda:



Ahora viene la segunda parte

rotaDerecha(Nodo); //paso el nodo 4

|  |
| --- |
| **struct Nodo \*rotaDerecha(struct Nodo \*y)**  {  struct Nodo \*x = y->izq;  struct Nodo \*T2 = x->der;  x->der = y;  y->izq = T2;  y->alto = max(alto(y->izq), alto(y->der))+1;  x->alto = max(alto(x->izq), alto(x->der))+1;  return x;  } |

Arrancamos con las dos primeras líneas

struct Nodo \*x = y->izq; //en y recibo el nodo 4

struct Nodo \*T2 = x->der;

ojo: T2 apunta a nulo

Seguimos con estas dos líneas de la función:

x->der = y;

y->izq = T2;

lo que me deja:



Y la función retorna x que finalmente se termina asignando como raíz. Quedaría perfectamente balanceado (redibujando):



Finalmente insertamos el nodo 1, lo que no nos va a dar problemas porque el árbol no quedará desbalanceado en ninguno de sus nodos.

