
**TECNOLÓGICO DE COSTA RICA**

**ÁREA ACADÉMICA DE INGENIERÍA EN COMPUTADORES**

**ALGORITMOS Y ESTRUCTURAS DE DATOS I**

**PROFESOR: ISAAC RAMÍREZ HERRERA**

# INVESTIGACIÓN: ÁRBOLES AVL

**ESTUDIANTES:**

**BRAYAN ALFARO GONZÁLEZ**

**JOSÉ JULIÁN CAMACHO HERNÁNDEZ**

**FÁTIMA LEIVA CHINCHILLA**

**JOSÉ FABIÁN MENDOZA MATA**

## Algoritmo de inserción AVL

El algoritmo para insertar un elemento en un árbol AVL, es similar al utilizado en un árbol binario convencional. Sin embargo, este difiere dado que en cada recursión se tiene que tomar en cuenta el factor de equilibrio de cada uno de los nodos y la altura en que cada uno se encuentra. Esto con el fin de verificar que efectivamente se trata de un árbol balanceado y no de un árbol normal, ya que los últimos pueden llegar a ser ineficientes dependiendo del orden de inserción de los elementos.

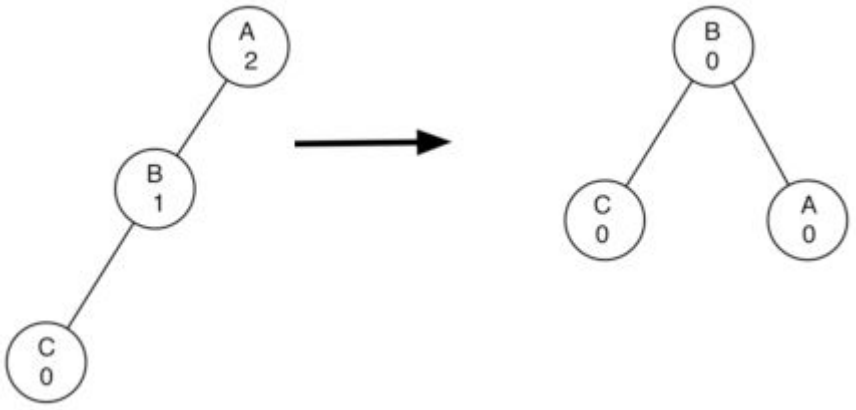
En el método; en primer lugar, se recorre el árbol hasta el punto donde al nodo le corresponde ser insertado. Esto siguiendo el algoritmo normal de un árbol binario, comparando el valor del nodo en cada recursión de la siguiente manera: si el valor del nodo a insertar es mayor que el nodo actual, se busca en el subárbol derecho, y si es menor en el izquierdo. Esto es un recorrido de exploración.

Posteriormente, al encontrar su posición, se hacen las verificaciones de desequilibrio, y se realizan algunas de las rotaciones correspondientes que pueden ser:

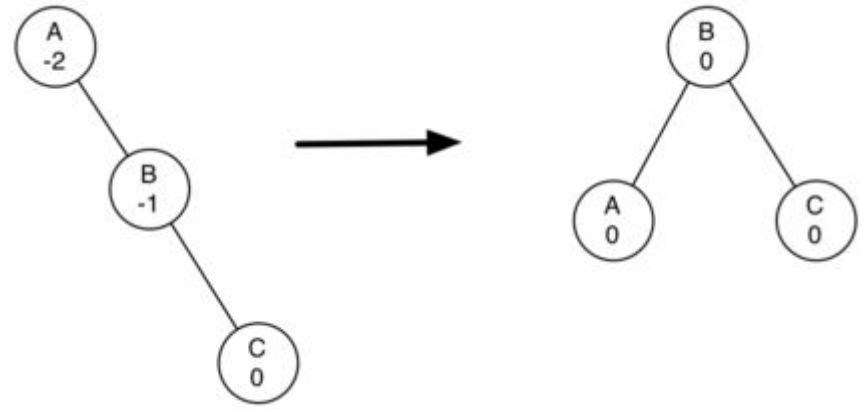
- Rotación derecha: se realiza un cambio en la posición de los nodos, de tal manera que el nodo central queda como la raíz del subárbol y quien era la raíz, queda como el nodo derecho de la nueva raíz.
- Rotación izquierda: de igual manera, en esta rotación se asigna la raíz al nodo central y la raíz anterior pasa a ser el nodo izquierdo de la nueva raíz.

Los desequilibrios pueden ser de 4 tipos:

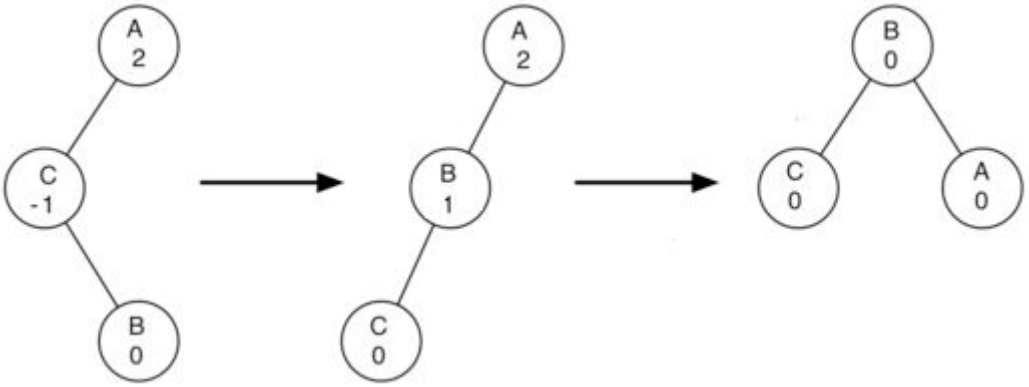
1. Caso izquierda - izquierda: se resuelve con una rotación derecha.



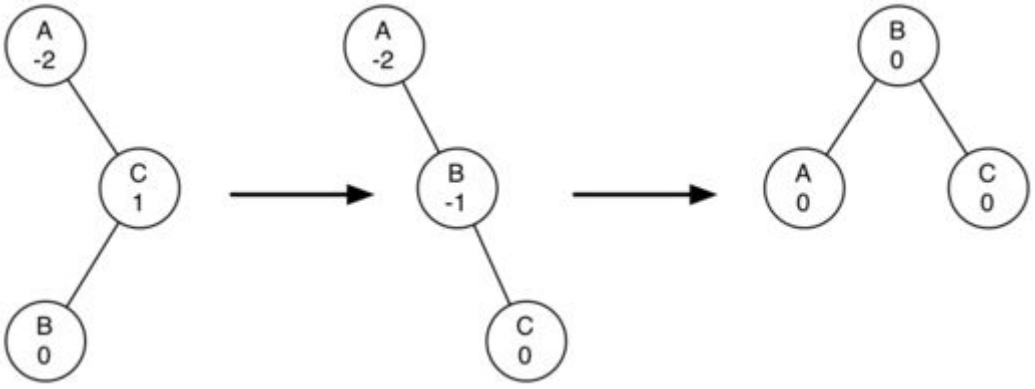

2. Caso derecha - derecha: se resuelve con una rotación izquierda.




3. Caso izquierda - derecha: se resuelve con una rotación izquierda, lo que resulta en un caso izquierda - izquierda, que se resuelve con rotación derecha.



4. Caso derecha - izquierda: de manera contraria al caso anterior, este se resuelve con una rotación derecha simple, y posteriormente una izquierda.




El siguiente fragmento presenta el código de cómo se insertaría un nodo en el árbol AVL. Como se ha mencionado, la inserción del nodo sería similar a la inserción en un árbol binario simple, con la diferencia de que cada vez que se inserta un nodo se debe revisar el factor de equilibrio y dependiendo de este valor se haría la rotación correspondiente.

```
7       public void insert(int e){
8           root=this.insert(e, root);
9       }
10      private Node insert(int e, Node current) {
11          if (current == null) {
12              return new Node(e);
13          } else if (e > current.element) {
14              current.setRigth(this.insert(e, current.rigth));
15          } else if (e < current.element) {
16              current.setRigth(this.insert(e, current.rigth));
17          } else {
18              return current;
19          }
20
```

```
45      private Node findMin(Node current){
46          if(current.getLeft()==null){
47              return current;
48          }else {
49              return findMin(current.left);
50          }
51      }
52      private Node rearrange(Node current) {
53          current.setBalance(getHeight(current.rigth) - getHeight(current.left));
54          int balance = current.getBalance();
55
56          if (balance > 1) {
57              if ( current.rigth.getBalance()>0) {
58                  return rotateDD(current);
59              } else {
60                  return rotateDI(current);
61              }
62          } else if (balance < 1) {
63              if (current.left.getElement()<0) {
64                  return rotateII(current);
65              } else {
66                  return rotateID(current);
67              }
68          } else {
69              return current;
70          }
71      }
```

Ahora aparecen las rotaciones a realizar una vez que el factor de equilibrio se ha "desequilibrado":

```
73      private int getHeight(Node current){
74          if(current==null){
75              return 0;
76          }else{
77              int leftHeight = this.getHeight(current.left);
78              int rigthHeight = this.getHeight(current.rigth);
79
80              return leftHeight>rigthHeight?1+leftHeight:1+rigthHeight;
81          }
82      }
83
84      private Node rotateDD(Node grandPa){
85          Node dad=grandPa.getRigth();
86          grandPa.setRigth(null);
87          this.insert(grandPa.getElement(),dad);
88          return dad;
89      }
90
91      private Node rotateII(Node grandPa){
92          Node dad=grandPa.getLeft();
93          grandPa.setLeft(null);
94          this.insert(grandPa.getElement(),dad);
95          return dad;
96      }
```

```
98      private Node rotateDI(Node grandPa){
99          Node dad=grandPa.getRigth();
100         Node child=dad.getLeft();
101
102         grandPa.setRigth(child);
103         dad.setLeft(null);
104         this.insert(dad.getElement(),child);
105         return this.rotateDD(grandPa);
106     }
107
108     private Node rotateID(Node grandPa){
109         Node dad=grandPa.getLeft();
110         Node child=dad.getRigth();
111
112         grandPa.setLeft(child);
113         dad.setRigth(null);
114         this.insert(dad.getElement(),child);
115         return this.rotateII(grandPa);
116     }
117 }
```

## Algoritmo de eliminación AVL

El algoritmo de eliminación de un nodo en un árbol AVL, consta de diversas etapas. En primer lugar, se realiza una exploración por el árbol hasta encontrar el nodo que se desea eliminar. Posteriormente, se realizan verificaciones para lograr conocer si el nodo es una hoja, o si tiene uno o dos hijos, y a partir de esta información se procede de la siguiente manera:

- Si es una hoja: simplemente se corta su referencia.
- Si tiene un solo hijo: el hijo toma la posición de su padre en el árbol.
- Si tiene dos hijos: se busca el valor mínimo a la derecha o el máximo a la izquierda para que tomen el lugar del padre, y luego recursivamente se elimina este nodo (el de valor mínimo o máximo, según se haya escogido).

Una vez terminado el proceso de eliminación, se recalculan los balanceos. Posteriormente, se realizan las transformaciones explicadas anteriormente (en el algoritmo de inserción) al encontrar disposiciones de balanceo no adecuadas en el árbol resultante.

El código es el siguiente:

```
24      public void remove(int e){
25          root=this.remove(e, root);
26      }
27
28      private Node remove(int e, Node current){
29          if(current==null){
30              return null;
31          }else if(e<current.getElement()){
32              current.setLeft(this.remove(e,current.left));
33          }else if(e>current.getElement()){
34              current.setRigth(this.remove(e,current.rigth));
35          }else if(current.getLeft()!=null && current.getRigth()!=null){
36              current.setElement(findMin(current.getRigth()).getElement());
37              current.setRigth(remove(current.getElement(),current.getRigth()));
38          }else{
39              current=current.getLeft()!=null?current.getLeft():current.getRigth();
40          }
41          this.rearrange(current);
42          return current;
43      }
```

Para acceder al video explicativo del algoritmo AVL puede ingresar al siguiente link:

https://youtu.be/6U9Jye7iP2E