Árbol balanceados

José Luis Quiroz Fabián

1 Árbol AVL

Un árbol AVL es un tipo especial de árbol binario ideado por los matemáticos rusos Adelson-Velskii y Landis. Fue el primer árbol de búsqueda binario auto-balanceable que se ideó. Los árboles AVL están siempre equilibrados de tal modo que para todos los nodos, la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha o viceversa. Debido al equilibrio en los árboles AVL, la complejidad de una búsqueda en uno de estos árboles se mantiene siempre en orden de complejidad *O(log n)* (igual al de la búsqueda binaria). Para mantener la propiedad de equilibrio en los árboles AVL, la inserción y el borrado de los nodos se realiza de una forma especial. Si al realizar una operación de inserción o borrado se rompe el equilibrio, se tiene que realizar una serie de rotaciones de los nodos a fin de mantener el equilibrio.

2 Rotaciones

2.0.1 Rotación Simple a la Izquierda (LL)

Esta rotación se usa cuando se inserta un nodo en el subárbol izquierdo del hijo izquierdo de un nodo desbalanceado. Para corregir el desbalance, se realiza una rotación hacia la derecha.

2.0.2 Rotación Simple a la Derecha (RR)

Esta rotación se usa cuando se inserta un nodo en el subárbol derecho del hijo derecho de un nodo desbalanceado. Para corregir el desbalance, se realiza una rotación hacia la izquierda.

2.0.3 Rotación Doble Derecha-Izquierda (RL)

Esta rotación se usa cuando se inserta un nodo en el subárbol izquierdo del hijo derecho de un nodo desbalanceado. Es una combinación de una rotación a la derecha seguida por una rotación a la izquierda.

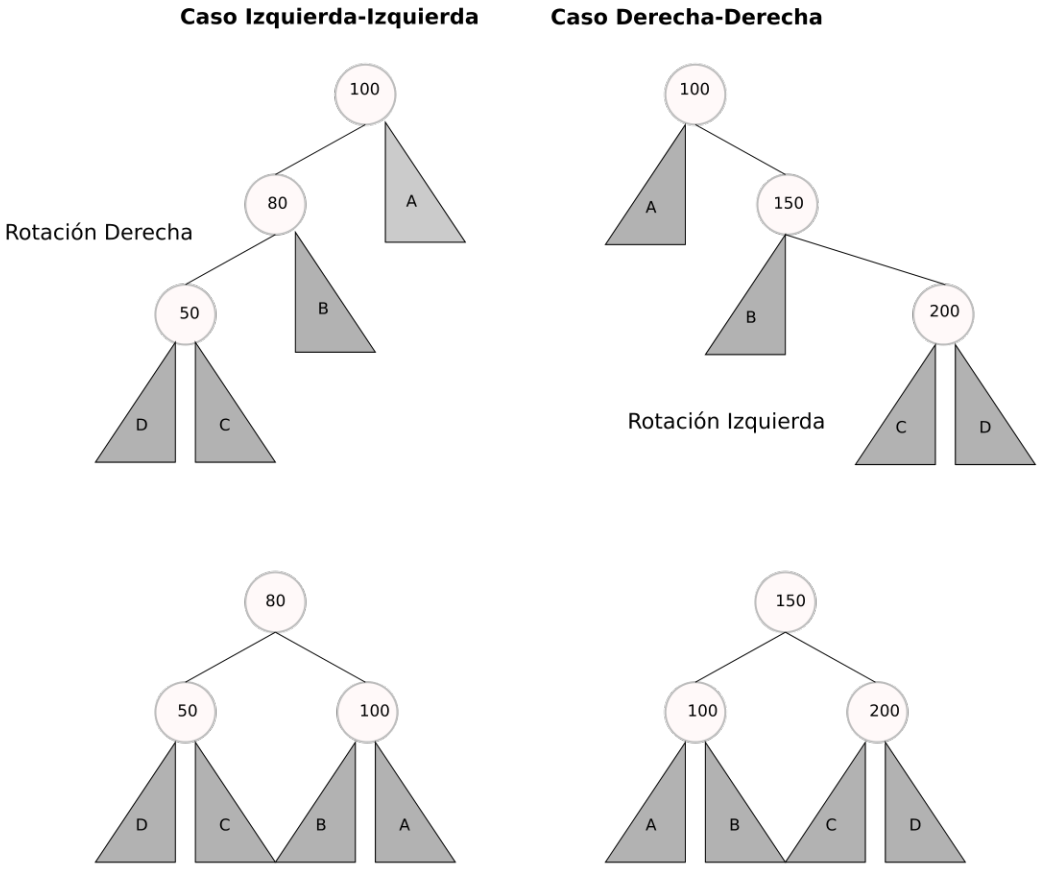

Figure 1: Rotaciones simples.

2.0.4 Rotación Doble Izquierda-Derecha (LR)

Esta rotación se usa cuando se inserta un nodo en el subárbol derecho del hijo izquierdo de un nodo desbalanceado. Es una combinación de una rotación a la izquierda seguida por una rotación a la derecha.

3 Implementación

Una propuesta de implementación se muestra en el Código 1. El factor de equilibrio se puede obtener restando la altura del sub-árbol izquierdo menos la altura de sub-árbol derecho o viceversa, solo se debe cuidar que rotación usar.

```
1 public class ArbolAVL<E extends Comparable<E>> {
2
3     private class NodoAVL <T>{
4         E dato;
5         NodoAVL<T> izquierdo, derecho;
```

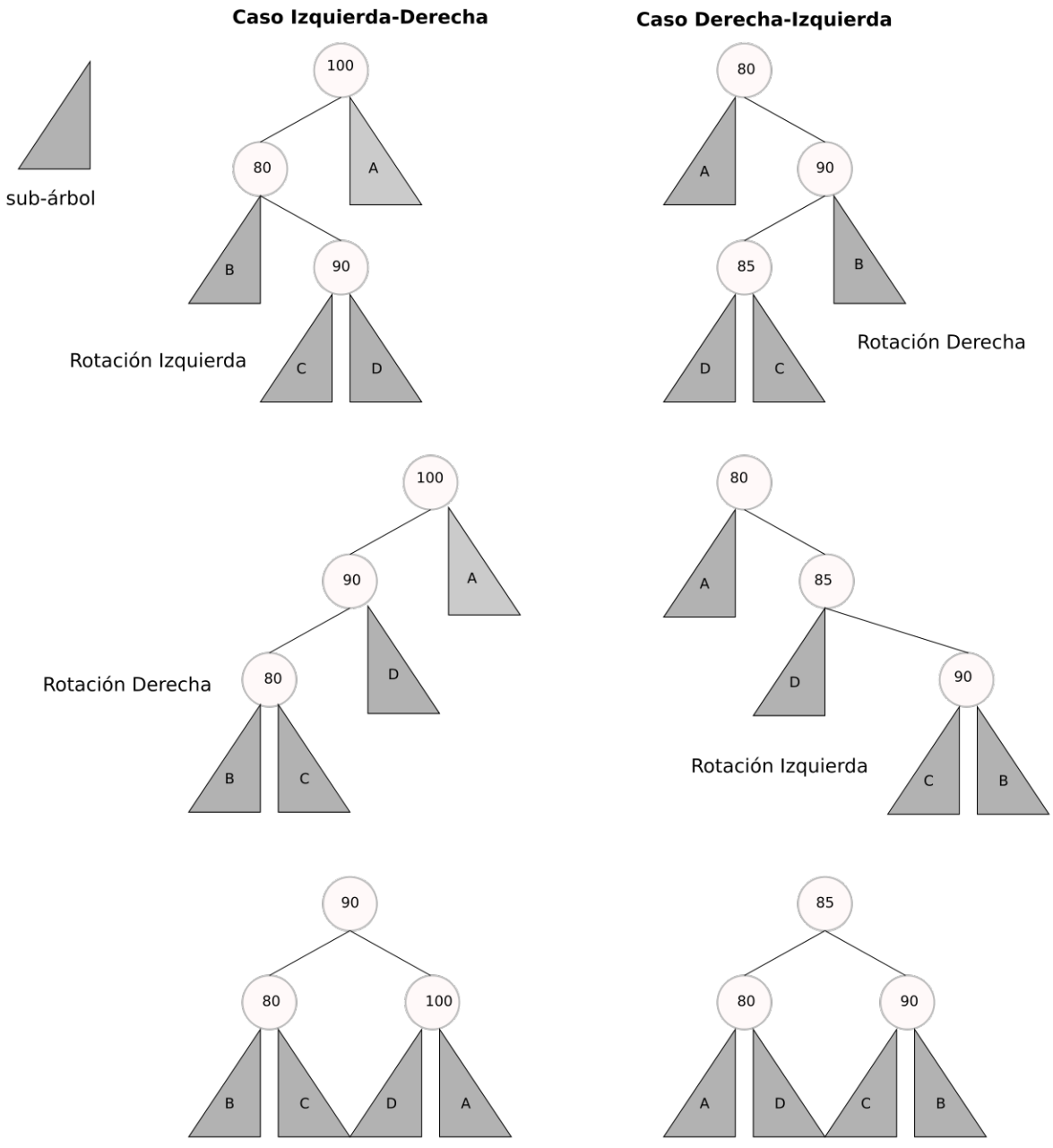

Figure 2: Rotaciones dobles.

```
6           int altura;
7
8           // Constructor
9           public NodoAVL(E d) {
10              dato = d;
11              altura = 1; // El nodo es un hoja
12          }
13      }
14
15      private NodoAVL<E> raiz;
16
17      // Metodo para obtener la altura de un nodo
18      int altura(NodoAVL<E> nodo) {
19          if (nodo == null)
20              return 0;
21          return nodo.altura;
22      }
23
24      // Metodo para calcular el factor de equilibrio de un nodo
25      int obtenerFactorEquilibrio(NodoAVL<E> nodo) {
26          if (nodo == null)
27              return 0;
28          return altura(nodo.izquierdo) - altura(nodo.derecho);
29      }
30
31      // Rotaciones
32      NodoAVL<E> rotarDerecha(NodoAVL<E> y) {
33          NodoAVL<E> x = y.izquierdo;
34          NodoAVL<E> T2 = x.derecho;
35
36          // Realizar rotacion
37          x.derecho = y;
38          y.izquierdo = T2;
39
40          // Actualizar alturas
41          y.altura = Math.max(altura(y.izquierdo), altura(y.derecho))
        + 1;
42          x.altura = Math.max(altura(x.izquierdo), altura(x.derecho))
        + 1;
43
44          // Devolver la nueva raiz
45          return x;
46      }
47
48      NodoAVL<E> rotarIzquierda(NodoAVL<E> x) {
49          NodoAVL<E> y = x.derecho;
50          NodoAVL<E> T2 = y.izquierdo;
51
52          // Realizar rotacion
53          y.izquierdo = x;
54          x.derecho = T2;
55
56          // Actualizar alturas
57          x.altura = Math.max(altura(x.izquierdo), altura(x.derecho))
        + 1;
58          y.altura = Math.max(altura(y.izquierdo), altura(y.derecho))
        + 1;
```

```
59
60          // Devolver la nueva raiz
61          return y;
62      }
63
64      // Metodo para insertar un dato E en el arbol
65      NodoAVL<E> insertar(NodoAVL<E> nodo, E dato) {
66          /* 1. Realizar la insercion normal de ABB */
67          if (nodo == null)
68              return (new NodoAVL<E>(dato));
69
70          if (dato.compareTo(nodo.dato)<0)
71              nodo.izquierdo = insertar(nodo.izquierdo, dato);
72          else if (dato.compareTo(nodo.dato)>0)
73              nodo.derecho = insertar(nodo.derecho, dato);
74          else // Los datos duplicados no se permiten en el arbol AVL
75              return nodo;
76
77          /* 2. Actualizar la altura del nodo ancestro */
78          nodo.altura = 1 + Math.max(altura(nodo.izquierdo),
79                                     altura(nodo.derecho));
80
81          /* 3. Obtener el factor de equilibrio de este nodo ancestro
82             para verificar si se desequilibro */
83          int balance = obtenerFactorEquilibrio(nodo);
84
85          // Si el nodo se desequilibra, hay 4 casos
86
87          // Rotacion simple a la derecha
88          if (balance > 1 && dato.compareTo(nodo.izquierdo.dato)<0)
89              return rotarDerecha(nodo);
90
91          // Rotacion simple a la izquierda
92          if (balance < -1 && dato.compareTo(nodo.derecho.dato)>0)
93              return rotarIzquierda(nodo);
94
95          // Rotacion izquierda-derecha
96          if (balance > 1 && dato.compareTo(nodo.izquierdo.dato)>0) {
97              nodo.izquierdo = rotarIzquierda(nodo.izquierdo);
98              return rotarDerecha(nodo);
99          }
100
101         // Rotacion derecha-izquierda
102         if (balance < -1 && dato.compareTo(nodo.derecho.dato)<0) {
103             nodo.derecho = rotarDerecha(nodo.derecho);
104             return rotarIzquierda(nodo);
105         }
106
107         /* devolver el apuntador del nodo (sin cambios) */
108         return nodo;
109     }
110
111
112
113     // Metodo para iniciar la insercion y mantener la raiz
        actualizada
114     public void insertar(E dato) {
```

```
115         raiz = insertar(raiz, dato);
116     }
117
118
119     public void mostrarArbol() {
120         mostrarArbol(raiz, 0);
121     }
122
123
124     private void mostrarArbol(NodoAVL<E> nodo, int nivel) {
125         if (nodo == null) {
126             return;
127         }
128
129         mostrarArbol(nodo.derecho, nivel + 1);
130         System.out.println("  ".repeat(nivel * 2) + nodo.dato);
131         mostrarArbol(nodo.izquierdo, nivel + 1);
132     }
133
134 }
```

Código 1: Propuesta de implementación de un árbol AVL

4 Prueba del Árbol AVL

El Código 2 muestra una propuesta de clase **App** para verificar la implementación del árbol AVL.

```
1
2 public class App {
3
4
5   public static void main(String[] args){
6
7
8         ArbolAVL<Integer> avl= new ArbolAVL<Integer>();
9
10
11        avl.insertar(20);
12        avl.insertar(4);
13        avl.insertar(5);
14        avl.insertar(10);
15        avl.insertar(9);
16        avl.insertar(15);
17        avl.insertar(25);
18
19        avl.mostrarArbol();
20
21        System.out.println("\n");
22
23        ArbolAVL<Integer> avl2= new ArbolAVL<Integer>();
24
25
26        avl2.insertar(9);
27        avl2.insertar(5);
```

```
28          avl2.insertar(11);
29          avl2.insertar(10);
30          avl2.insertar(15);
31          avl2.insertar(2);
32          avl2.insertar(7);
33          avl2.insertar(0);
34          avl2.insertar(6);
35          avl2.insertar(8);
36          avl2.insertar(13);
37          avl2.insertar(20);
38          avl2.insertar(14);
39          avl2.mostrarArbol();
40
41      }
42
43  }
```

Código 2: Prueba AVL

5 TreeMap

TreeMap en Java es una implementación de la interfaz Map basada en un árbol rojo-negro. Es una parte de las Collections Framework y proporciona una eficiente manera de almacenar pares clave-valor.

5.1 Estructura Interna

La estructura interna de un TreeMap es un árbol rojo-negro[1], que es una forma de árbol binario de búsqueda auto-balanceado. Cada nodo en el árbol tiene un color (rojo o negro) y enlaces a su nodo padre, hijo izquierdo y hijo derecho. Los elementos en un TreeMap se almacenan en un orden natural basado en las claves o un Comparator.

5.2 Valores

Almacena los datos en pares de clave-valor donde cada clave es única. Si se intenta insertar una clave que ya existe, su valor asociado se sobrescribirá con el nuevo valor.

5.3 Operaciones Comunes

- Inserción (put): Para insertar un nuevo par clave-valor, TreeMap ubica el lugar correcto del nuevo nodo en el árbol y lo inserta como un nodo rojo o

[1]Un árbol rojo-negro es un árbol binario de búsqueda equilibrado. La estructura original fue creada por Rudolf Bayer en 1972, que le dio el nombre de árboles-B binarios simétricos, pero tomó su nombre moderno en un trabajo deLeo J. Guibas y Robert Sedgewick realizado en 1978. Puede buscar, insertar y borrar en un tiempo O(log n), donde n es el número de elementos del árbol.

negro según las reglas del árbol rojo-negro para mantener el balanceo del árbol.

- Búsqueda (get): Para buscar un valor asociado a una clave, TreeMap realiza una búsqueda binaria en el árbol, comparando la clave objetivo con las claves en los nodos visitados, navegando hacia la izquierda o la derecha hasta encontrar la clave o determinar que no está presente.
- Eliminación (remove): La eliminación de un nodo puede ser más compleja, ya que después de eliminar el nodo, TreeMap puede necesitar reestructurar el árbol y reorganizar los colores de los nodos para mantener las propiedades del árbol rojo-negro.
- Navegación: TreeMap proporciona varias formas de navegar a través del mapa, como firstEntry, lastEntry, higherEntry, lowerEntry, entre otros, permitiendo recorridos eficientes y búsquedas específicas en el árbol.

```
1
2  public class Palabra implements Comparable<Palabra>{
3
4   String palabra;
5   String significado;
6   public Palabra(String palabra, String significado) {
7     super();
8     this.palabra = new String(palabra);
9     this.significado = new String(significado);
10  }
11  public String getPalabra() {
12    return palabra;
13  }
14  public void setPalabra(String palabra) {
15    this.palabra = palabra;
16  }
17  public String getSignificado() {
18    return significado;
19  }
20  public void setSignificado(String significado) {
21    this.significado = significado;
22  }
23  @Override
24  public int compareTo(Palabra o) {
25
26    return palabra.compareTo(o.getPalabra());
27
28  }
29  public String toString(){
30
31    return palabra;
32
33  }
34
35 }
36
37 import java.util.LinkedList;
```

```
38 import java.util.TreeMap;
39
40
41 public class Principal {
42   public static void  main(String[] args){
43
44     Integer x;
45
46     TreeMap<Integer,Integer> arbol = new TreeMap<Integer,Integer>()
       ;
47
48     arbol.put(1, 1);
49     arbol.put(2, 2);
50     arbol.put(3, 3);
51     arbol.put(4, 4);
52     arbol.put(5, 5);
53     x = arbol.get(2);
54     System.out.println(x);
55
56     TreeMap<Integer,Palabra> arbol2 = new TreeMap<Integer,Palabra
       >();
57
58     arbol2.put(1, new Palabra("Hola","Es un simple saludo"));
59     arbol2.put(2, new Palabra("Barco","Medio de transporte"));
60     arbol2.put(3, new Palabra("Auto","Medio de transporte"));
61     arbol2.put(4, new Palabra("Avion","Medio de transporte"));
62
63     System.out.println(arbol2.get(3));
64
65
66     TreeMap<Integer,LinkedList<Integer>> arbol3 = new TreeMap<
       Integer,LinkedList<Integer>>();
67
68     LinkedList<Integer> aux;
69
70     for(int i=0;i<5;i++){
71
72       aux = arbol3.get(1);
73       if(aux!=null){
74         System.out.println("Si esta el elemento");
75         aux.addLast(1);
76       }
77       else{
78         System.out.println("No esta el elemento");
79         aux = new LinkedList<Integer>();
80         aux.addLast(1);
81         arbol3.put(1, aux);
82       }
83
84     }
85   }
86
87
88 }
```

Para iterar sobre un TreeMap se debe indicar si es sobre las entradas, las llaves o los valores.

```
 Iterar sobre las Entradas (Key-Value Pairs)


TreeMap<Integer, String> treeMap = new TreeMap<>();
// Suponer que treeMap esta inicializado y lleno de pares clave-
    valor

for (Map.Entry<Integer, String> entry : treeMap.entrySet()) {
    System.out.println("Key = " + entry.getKey() + ", Value = " +
    entry.getValue());
}

Iterar solo sobre las Claves
for (Integer key : treeMap.keySet()) {
    System.out.println("Key = " + key);
}


Iterar solo sobre los Valores
for (String value : treeMap.values()) {
    System.out.println("Value = " + value);
}
```

6 Ejercicios

- Crear/Abrir un proyecto en **Visual Studio Code** llamado **Edatos**. Crear el paquete **edatos.nolineales.arboles.binario.abb.avl**. En el paquete almacenar las clases **ArbolAVL** y **App**. Probar su implementación.
- Implemente el ejemplo de TreeMap en el paquete **edatos.nolineales.arboles.binario.abb.treemap**