EDA I

Práctica 02
Ejercicio 2 AVLTree

Juan Antonio Rodríguez Baeza

```
/**
* Esta clase es un árbol binario equilibrado que implementa el árbol AVL interfaz Collection
* métodos de rotación individuales y dobles.
*/
public class AVLTree<T>
implements Collection<T>, Iterable<T>,
BinarySearchTree<T>{

/**
* La clase tiene los siguientes atributos: una referencia al nodo raíz, el numero de
* elementos en el árbol y un atributo transitorio modCount para verificar que es en un
* estado coherente
*/
private AVLNode<T> root;
private int treeSize;
private int modCount;

/**
* Método privado para borrar el árbol con una exploración postorden de los nodos
*/
private void deleteTree(AVLNode<T> t){…}

/**
* Método privado para encontrar un objeto dentro del árbol
*/
private AVLNode<T> findNode(Object item)


/**
* Constructor por defecto
*/
public AVLTree()


/**
* Método publico que añade haciendo uso del método privado addNode, el elemento
* especificado a este árbol si no está ya presente.
*/
public boolean add(T item)

/**
* Método privado que realiza la acción de añadir.
*/
private AVLNode<T> addNode(AVLNode<T> t, T item)
```

```java
/**
* Método público que elimina, haciendo uso del método privado remove, un objeto del árbol
*/
public boolean remove(Object item){…}

/**
* Método privado que realiza la acción de eliminar.un objeto del interior de un nodo
*/
private AVLNode<T> remove(AVLNode<T> t, T item){…}

/**
* Método privado que realiza la acción de eliminar un nodo.
*/
private AVLNode<T> removeNode(AVLNode<T> removalNode){…}


/**
* Método privado que realiza la acción de eliminar el menor de los elementos del árbol.
*/
private AVLNode<T> removeMin(AVLNode<T> t){…}


/**
* Método privado que realiza la búsqueda del menor de los elementos del árbol.
*/
private AVLNode<T> findMin(AVLNode<T> t){…}

/**
* Método privado que devuelve el peso del nodo.
*/
private static <T> int height(AVLNode<T> t){…}

/**
* Método privado que devuelve el mayor de dos enteros.
*/
private static int max(int a, int b){…}

/**
* Método privado que realiza una rotación simple a la derecha sobre el nodo que se le pasa por
* parámetro.
*/
private static <T> AVLNode<T> singleRotateRight(AVLNode<T> p){…}
```

```
/**
* Metodo privado que realiza una rotación simple a la izquierda sobre el nodo que se le pasa por
* parámetro
*/
private static <T> AVLNode<T> singleRotateLeft(AVLNode<T> p){…}

/**
* Método privado que realiza una rotación doble a la derecha sobre el nodo que se le pasa por
* parámetro.
*/
private static <T> AVLNode<T> doubleRotateRight(AVLNode<T> p){…}

/**
* Método privado que realiza una rotación doble a la izquierda sobre el nodo que se le pasa por
* parámetro.
*/
private static <T> AVLNode<T> doubleRotateLeft(AVLNode<T> p){…}

/**
* Método publico para limpiar el árbol, hace uso del método privado deleteTree, manteniendo
* la referencia al nodo raíz
*/
public void clear(){…}

/**
* Método público que, haciendo uso del método findNode, busca el nodo que le pasen por
* parámetro y devuelve true si el árbol contiene el nodo, en caso contrario devuelve false.
*/
public boolean contains(Object item){…}

/**
* Devuelve true si el árbol no contiene elementos, es decir, que esté vacío.
*/
public boolean isEmpty(){…}

/**
* Devuelve un iterador sobre los elementos de este árbol. Los elementos se devuelven en orden
* ascendente.
*/
public Iterator<T> iterator(){…}

/**
* Devuelve el número de elementos en este árbol.
*/
public int size(){…}
```

```
/**
* Devuelve una matriz que contiene todos los elementos de este árbol. El orden de los
* elementos de la matriz lo da el iterador de elementos en el árbol
*/
public Object[] toArray(){…}

/**
* Devuelve una representación de cadena de este árbol. La representación es una lista separada
* por comas con el fin iterador entre corchetes .
*/
public String toString(){…}

/**
* Busca el elemento especificado en el árbol y devuelve el valor del nodo que coincide elemento
* como clave .
*/
public T find(T item)


/**
* Adicionalmente la implementación incorpora las subclases TreeIterator y AVLNode
*/
private class TreeIterator implements Iterator<T>{…}

private static class AVLNode<T>{…}

}// Fin de la clase
```

Si comparamos las estructuras AVLTree con las ArrayList, las diferencias las encontramos en el orden de complejidad, puesto que el coste algorítmico de una ArrayList, en el peor de los casos es $O(n)$, mientras que el AVLTree es $O(log)$.

Ventajas e inconvenientes de la implementación algorítmica del AVLTree.

La notable mejora del AVLTree es que la estructura se balancea para corregir las alturas de las ramas del árbol de manera que nunca haya demasiada diferencia de unas con respecto a otras. Además de, por ser un árbol binario de búsqueda, es una estructura ordenada.

El inconveniente es que al realizar operaciones, sobre todo eliminar, el coste crece, puesto que en el peor de los casos debemos recorrer la rama que necesita balanceo desde un nodo hoja hasta la raíz, para asegurarnos de que el balanceo es correcto.