# PRACTIA 2 EJERCICIO 2: ARBOLES AVLTREE

Juan Miguel Herrada Acosta

Juan Miguel Herrada Acosta
Grado en ingeniería informática. Estructura de datos y algoritmos I

Adicionalmente, se ha proporcionado todo lo relativo a la implementación del AVLTree, por lo que se pide que se EXPLIQUE en un documento (memoria) toda implementación suministrada, prestando especial atención a funciones principales de dicha estructura de datos como: add, remove, clear, contains, isEmpty, así como al uso propuesto del iterador. También se pide, en dicha memoria, que se enumeren las ventajas e inconvenientes de la resolución de este problema mediante el uso de estructuras arbóreas (AVLTree) en lugar de colecciones lineales (ArrayList) como se hizo la práctica 1 (Ejercicio 02). Razone y justifique, adecuadamente, cada una de las ventajas e inconvenientes.

Métodos:

- public boolean add (T item): añade un elemento al árbol si el elemento ya se encuentra dentro del árbol devolverá falso y si lo pudo añadir devolverá verdadero.
  Internamente igualamos raíz al resultado del método addNode al cual le pasamos la raíz y el elemento a añadir. Si en algún caso salta la excepción devolveremos falso, en caso contrario incrementaremos en uno el valor del tamaño del árbol, el contador de consistencia del mismo y devolveremos verdadero.

- public boolean remove (T item): elimina un elemento del árbol si el árbol contiene dicho elemento lo elimina y devuelve verdadero, si no lo contiene devuelve falso.

  Internamente igualamos raíz al resultado del método remove al cual le pasamos la raíz y el elemento a eliminar. Si en algún caso salta la excepción devolveremos falso, en caso contrario reducirá en uno el valor del tamaño del árbol, el contador de consistencia del mismo y devolveremos verdadero.

- public void clear (): elimina todos los elementos del árbol dejándolo vacío.
  Internamente incrementa en uno el valor del contador de consistencia, iguala a cero el tamaño del árbol e iguala el nodo raíz a null.

- public boolean contains (T item): devuelve verdadero si nuestro árbol contiene dicho elemento y falso en caso contrario.
  Internamente hace una llamada al método find (este método devuelve el nodo que contiene ese elemento o null si el elemento no se encuentra en el árbol), si el valor devuelto es igual a null el método devolverá falso y en caso contrario devolvería verdadero.

- public boolean isEmpty (): comprueba si el árbol está vacío o no.
  Internamente comprueba el tamaño del árbol es igual a cero, en tal caso devolveríamos verdadero y en caso contrario falso.

- public Iterator<T> iterator (): devuelve un iterador genérico de tipo T. Internamente devuelve la creación de un TreeIterator () este método es el constructor de la clase iterator interna de BSTree.
  Esta clase es necesario porque la necesidad de un iterador para recorrer la colección.
  TreeIterator () internamente igualamos el siguiente nodo a raíz y comprobamos si el siguiente nodo es distinto de null, cuando sea distinto de null entraremos a comprobar que mientras el hijo izquierdo del siguiente nodo sea distinto de null igualaremos el siguiente nodo a su hijo izquierdo.

**Ventajas:**

- El coste de recorrer BSTree es $O(\log n)$ comparado con las lineales que sería de $O(n)$ $o$ $O(n * \log n)$.
- La inserción de valores ya tiene un orden, es decir, no hace falta ordenar la colección.
- El árbol no se degenera ya que tiene un parámetro para su control y métodos para su corrección.

**Inconvenientes:**

- El coste de recalcular el orden y mover las nodos dependiendo del factor de ramificación el cual seria $O(\log n)$.