EDA I

Práctica 02
Ejercicio 1

Juan Antonio Rodríguez Baeza

```
/**
*  Esta clase implementa la interfazes Collection, Iterable, Cloneable y
*  BinarySearchTree; utilizando un árbol de búsqueda binario
*  como la estructura de almacenamiento subyacente.
*/

public class BSTree<T>
implements Collection<T>, Iterable<T>, Cloneable,
BinarySearchTree<T>{
/**
* La clase dispone de tres atributos privados un BSTBode<T> para la raíz del árbol; un
* treesize para el tamaño del árbol; y un atributo transitorio modCount para verificar
* que es en un estado coherente
*/

private BSTNode<T> root;
private int treeSize;
transient private int modCount;

/**
* Método privado utilizado por remove ( ) y el iterador remove () para eliminar un nodo
* Le llega un nodo como parámetro, este es el nodo a eliminar, el método relaciona al
* padre de este nodo con su "nieto", y descarta el nodo.
*/
private void removeNode(BSTNode<T> dNode){…}

/**
* findNode recibe objeto en general que buscará a lo largo de la estructura del árbol,
* devolviendo una referencia al nodo que contiene el objeto, o null si la búsqueda es
* insatisfactoria
*/
private BSTNode<T> findNode(Object item) {…}

/**
*  Constructor por defecto. Crea un árbol binario de búsqueda vacío. Todos los
* elementos insertados en el árbol deben implementar la interfaz comparable.
*/
public BSTree() {…}

/**
* El metodo add se asegura de que el árbol contiene el elemento pasado por parámetro.
* Devuelve true si la operación inserta el nuevo elemento, y devuelve falso si el árbol ya
* contiene el elemento especificado y no permite duplicados.
*/
```

```
public boolean add(T item){…}

/**
* Elimina todos los elementos del árbol. El árbol quedarávacío.
*/
public void clear(){…}

/**
* Devuelve true si la colección contiene el elemento y falso en caso contrario. Para
* hacer esto buscara el nodo pasándoselo al metodo privado findNode.
*/
public boolean contains(Object item){…}

/**
* Devuelbe el resultado booleano de comparar el tamaño del árbol a 0, lo que retornara
* verdadero si esta vacio, y falso si contiene algún elemento.
*/
public boolean isEmpty(){…}

/**
* Devuelve un iterador sobre los elementos del arbol
*/
public Iterator<T> iterator(){…}

/**
* Método público para eliminar, a este método podrán referirse las clases que hagan uso
* de esta implementación, este método llama al método privado que es el que realmente
* se encarga de la eliminación.
*/
public boolean remove(Object item){…}

/**
* Devuelve el tamaño del árbol, retornando su atributo treeSize
*/
public int size(){…}

/**
* Devuelve un array que contiene todos los elementos de este árbol. El orden de los
* elementos de la array es el orden del 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 en orden del iterador y entre corchetes.
*/
public String toString(){…}

/**
* Método público para buscar, a este método podrá referirse cualquier clase que haga
* uso de esta implementación este método llama al método privado que es el que
* realmente se encarga de la búsqueda.
*/
public T find(T item){…}

/**
* Este método público llama al privado copyTree, para devolver una copia exacta del
* árbol.
*/
public Object clone(){…}

/**
* Utiliza el método estático copyTree() de la clase BinaryTree como modelo para crear
* un duplicado del árbol con t como raíz y devuelve una referencia a la raíz del árbol de
* copiado
*/
private static <T> BSTNode<T> copyTree(BSTNode<T> t){…}

/**
* Devuelve una cadena que muestra los elementos en el árbol binario utilizando la
* preorden ( NLR ) de exploración. La descripción es una lista de elementos separados
* por dos espacios en blanco.
*/
private static <T> String preorderDisplay(BSTNode<T> t){…}

/**
* Método público para mostrar el contenido del árbol en preorden, a él podrán
* referirse las clases que hagan uso de esta implementación, este método llama al
* método privado que es el que realmente se encarga de la operación a realizar.
* Finalmente devuelve un string con la cadena que ha calculado el método privado.
*/
public String preorderDisplay(){…}
```

```
/**
* Método para generar una cadena con una estructura en forma de árbol tumbado en el
* que se puede observar de manera mas comoda la disposición de los nodos.
*/
private static <T> String displayTree(BSTNode<T> t, int level){…}

/**
* Método publico que devuelve la cadena ordenada en forma de árbol del metodo
* privado, las clases que hagan uso de esta implementación podrán llamar a este
* metodo, que a su vez llamara al privado que se encarga de la realización de dicha
* cadena. Este metodo lo único que hace es devolver el string.
*/
public String displayTree(){…}


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

private static class BSTNode<T>{…}

}// Final de la clase
```

Ventajas e inconvenientes de la implementación algorítmica del BSTree.

La mayor ventaja radica en la potencial disminución del coste de las actuaciones (inserción, búsqueda eliminación, etc.) que en el mejor de los casos llega a ser logarítmica $O(log)$. Además la estructura siempre esta ordenada.

El inconveniente es que no se balancea, es decir, podemos encontrarnos con una estructura que solo contenga hijos izquierdos en cada nodo, por ejemplo, de manera que degenera en un ArrayList o en una LinkedList; con lo que pierde la ventaja del coste logarítmico de las actuaciones.