Implementación de un Árbol Binario de Búsqueda

Elaborado por: Arnaldo Quintero Segura.

Documentación Técnica

Para la implementación de este Árbol Binario de Búsqueda (ABB) se procederá a desarrollar cada una de las operaciones que se puede realizar utilizándola. Y una ligera explicación de por qué es de esta manera. Esta implementación de árbol binario soporta unicamente elementos de tipo entero, dado que fue lo visto en clase.

Antes, veamos la estructura de los nodos que componen cada elemento presente en el ABB. Esta sería la clase Node, llamada así por la palabra en inglés para Nodo:

Node

Esta clase posee tres atributos: El valor del nodo, de tipo genérico `<T>`, y los hijos derecho e izquierdo del nodo. De tipo `Node<T>`.

Estos atributos poseen cada uno su `getter` y su `setter` respectivamente. Siendo los atributos `value`, `left` y `right`. Además, la clase posee un método `children` que devuelve la cantidad de hijos que posee actualmente el Nodo. Método que es utilizado posteriormente en llamadas internas de las funciones del ABB.

Propiedades del ABB

Esta clase de Árbol Binario de Búsqueda, se llama `BinarySearchTree`, y contiene inicialmente un solo atributo. El apuntador a su nodo raíz, siendo este `root` de tipo `Node<Integer>`. Atributo que es completamente invisible a los ojos del usuario final, dada la implementación del ABB.

Listemos ahora a ver las operaciones que soporta nuestro árbol binario de búsqueda:

Insertar un valor

Para insertar un valor, el ABB posee el método `insert`, el cual recibe como parámetro el valor entero a ser insertado en el árbol. Y devuelve un booleano que nos indica si la operación ha sido exitosa o no. `true` en caso de que se haya insertado de manera correcta, `false` de manera contraria.

Para el proceso de inserción se posee el método interno `insert`, que recibe como parámetros un nodo, y el valor a insertar. De manera que se encarga de recursivamente buscar el nodo correcto para el cual debe insertar el valor. Siguiendo las condiciones que debe cumplir un ABB.

Este método devuelve un Nodo nuevo, de manera que el caso de la recursividad se detiene cuando se encuentra con un hijo de valor `null` e inserta el nuevo nodo en dicha posición. Retornando siempre el mismo nodo que fue pasado inicialmente, pero con el nuevo valor correctamente insertado.

En caso de que el valor se encuentre ya insertado en el árbol. El valor no se ve modificado.

Eliminar un valor

Para eliminar un valor de un ABB, tenemos la función `delete`, que recibe como parámetro el valor que se desea eliminar del ABB. En caso de que el valor no se encuentre en el árbol, el ABB no se ve modificado.

Este método utiliza internamente, uun método privado `delete` el cual recibe como parámetro el valor a eliminar, y el nodo a buscar recursivamente hasta conseguir el valor a eliminar. Y retorna el nodo que fue introducido, pero con el nodo deseado eliminado. Se pueden presentar tres opciones para eliminar un nodo del ABB:

El nodo a eliminar no tiene hijos

Por lo tanto, se procede a eliminar de manera tranquila del árbol, eliminando la referencia de su padre a él. Se retorna `null`.

El nodo a eliminar tiene un hijo

En este caso, se retorna el valor del hijo, de manera que la referencia del nodo desaparece, y se conectan el padre del eliminado con el hijo del eliminado.

El nodo a eliminar tiene dos hijos

Este es el caso más complejo, para el cual se debe buscar el sucesor del nodo a eliminar, ponerlo en la posición del nodo a eliminar. Y eliminar del sub-árbol que poseía al sucesor, el árbol derecho, a este nodo sucesor. Veamos en detalle en que consiste:

Este proceso consiste primero en buscar el sucesor, siendo este el hijo más izquierdo del sub-árbol derecho del nodo a eliminar. Para esto tenemos la función `mostLeftChild`, que dado un Nodo, busca el hijo más izquierdo de dicho nodo.

Al tener el sucesor. Se copia el valor en el nodo a eliminar. Y recursivamente se asigna al valor del sub-árbol derecho del nodo que ya fue cambiado, ese mismo nodo, pero eliminando el nodo sucesor.

De esta manera, recursivamente se elimina el nodo que se desea, y se pasa a eliminar el nodo sucesor luego de haber sido clonado.

Recorrido del árbol

Para realizar el recorrido a el árbol, se poseen tres métodos: `preOrder`, `inOrder` y `postOrder`, los cuales recorren el árbol de manera pre, in y post orden respectivamente. Cada uno de estos métodos devuelve un objecto de tipo cola `Queue<Integer>`, el cual posee los elementos en el orden correcto.

Se decidió utilizar una cola para representar esto, dado que bajo cualquier recorrido, se quiere ir insertando en una lista al final de la misma. Y que el primer elemento a sacar sea el primero en ser insertado. Por lo tanto, bajo una política FIFO, se utiliza el TAD Cola, que cumple dicha política.

Cada uno de estos métodos funcionan utilizando métodos privados, los cuales recursivamente van recorriendo el árbol y van agregando a la cola los valores de los nodos que han sido recorridos, en el orden que corresponda.

Impresión del árbol

Para la impresión del árbol, tenemos dós métodos: `printTree` y `print`. El primero imprime a la salida estándar cada uno de los nodos del árbol. Donde la raíz no tiene prefijo. Y a partir de ahí se va generando un nivel de indentación para cada hijo. Donde además se especifica si es el hijo derecho (R) o izquierdo (L). Como se muestra a continuación:

```
+- 8             <- nodo raíz
L  +- 4          <- hijo izq. del nodo raíz (Nodo L)
L  L  +- 1       <- hijo izq. del nodo L1 (Nodo LL)
L  R  +- 6       <- hijo der. del nodo L (Nodo LR)
L  R  R  +- 7    <- hijo der. del nodo LR (Nodo LRR)
R  +- 10         <- hijo der. del nodo raíz (Nodo R)
R  R  +- 13      <- hijo der. del nodo R (Nodo RR)
```

En caso de un árbol vacío, esta función nos imprime `Empty Tree`.

La segunda función, nos imprime el árbol ya en recorrido. Para esto, se utilizan los métodos especificados en la sección anterior. De manera que se imprimen los elementos resultantes en la Cola, luego de realizar el recorrido al árbol. Para diferenciar que tipo de recorrido se quiere, esta función recibe un parámetro de tipo `String`, el cual define el tipo de recorrido. Los valores válidos son `pre`, `post` e `in`. De igual manera, si se introduce un valor `String` que no sea ninguno de los anteriores, se imprimirá por defecto el recorrido In Orden, dado que este recorrido te emite los valores de un ABB en orden creciente.

Prueba de funcionalidad

Para probar la funcionalidad de esta clase, se incluye el ejecutable `ArbolBinarioDeBusqueda.jar`, el cual utiliza la función `Main` del fichero `MainExample.java`. En el cual se muestran inserciones, impresiones, búsquedas y eliminaciones de distintos tipos sobre un ABB.