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, y al iterador iterator del tipo TreeIterator. Razone y justifique adecuadamente cada una de las ventajas e inconvenientes enumerados.

ADD

Insertar el nuevo elemento, cumpliendo la propiedad de menores a la izquierda, mayores a la derecha: siempre se inserta en las hojas

```
public boolean add(T item)
{
    // t is current node in traversal, parent the previous node
    BSTNode<T> t = root, parent = null, newNode;
    int orderValue = 0;

    // terminate on on empty subtree
    while(t != null)
    {
        // update the parent reference.
        parent = t;

        // compare item and the current node value
        orderValue = ((Comparable<T>)item).compareTo(t.nodeValue);

        // if a match occurs, return false; otherwise, go left
        // or go right following search tree order
        if (orderValue == 0)
            return false;
        else if (orderValue < 0)
            t = t.left;
        else
            t = t.right;
    }
```

```
        // create the new node
        newNode = new BSTNode<T>(item,parent);

        if (parent == null)
            // this is the first node added. make it root
            root = newNode;
        else if (orderValue < 0)
            // attach newNode as the left child of parent
            parent.left = newNode;
        else
            // attach newNode as the right child of parent
            parent.right = newNode;

        // increment the tree size and modCount
        treeSize++;
        modCount++;

        // we added a node to the tree
        return true;
    }
```

Si el elemento a insertar se encuentra en el árbol, no se hace nada. En caso contrario, se inserta en el lugar donde acaba la búsqueda. La búsqueda acaba sin éxito al llegar a un subárbol izquierdo o derecho que está vacío.

Clear:

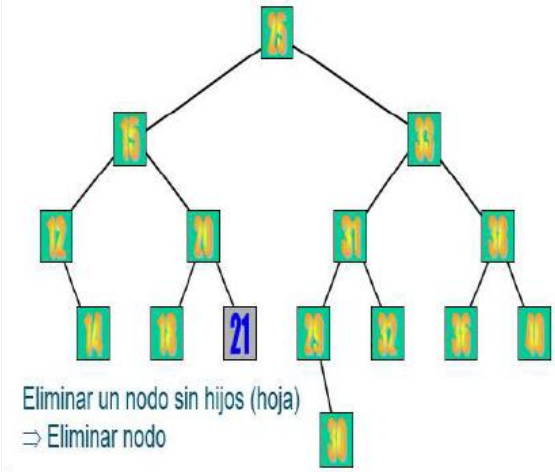
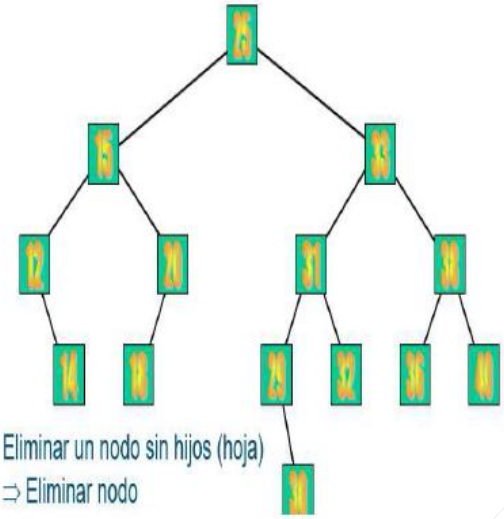
Elimina el árbol binario de búsqueda igualando el tree a null y el tamaño del árbol a 0.

```
public void clear()
{
    modCount++;
    treeSize = 0;
    root = null;
}
```

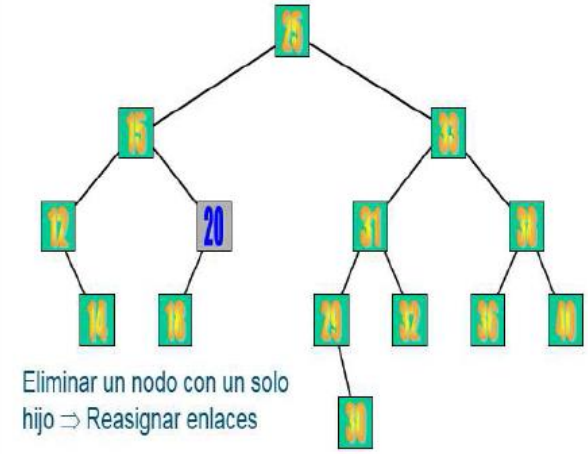
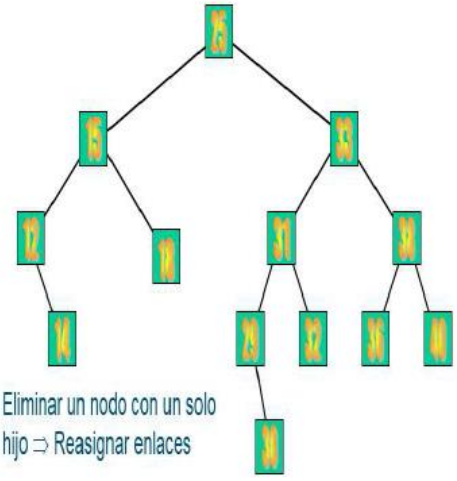
Remove:

Esta función realiza una búsqueda en el árbol, si encuentra el nodo a eliminar llamara a removeNode(). Si el elemento a eliminar o extraer se encuentra en el árbol, el comportamiento del algoritmo depende del número de hijos que tenga dicho nodo:

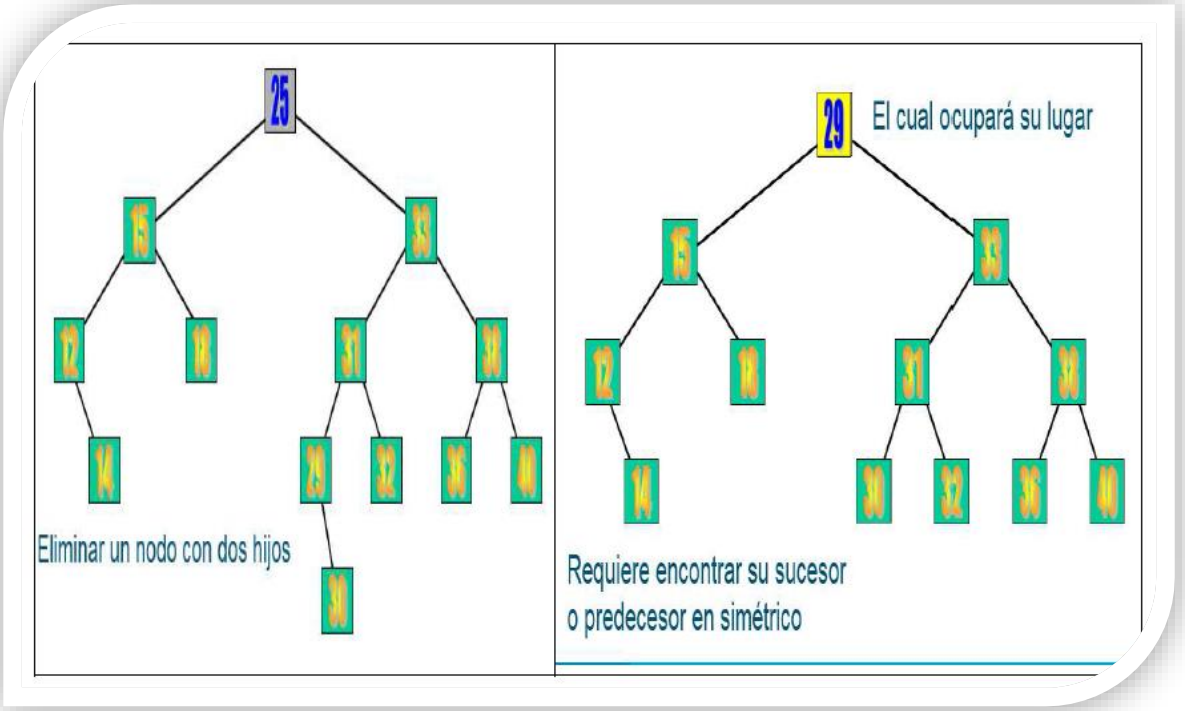
Eliminar nodo sin Hijos

Nodo a eliminar: 21	Estado del árbol tras la eliminación
	
Eliminar un nodo sin hijos (hoja) ⇒ Eliminar nodo	Eliminar un nodo sin hijos (hoja) ⇒ Eliminar nodo

Eliminar nodo con un solo hijo

Nodo a eliminar: 20	Estado del árbol tras la eliminación
	
Eliminar un nodo con un solo hijo ⇒ Reasignar enlaces	Eliminar un nodo con un solo hijo ⇒ Reasignar enlaces

Eliminar nodo con 2 hijos:



Contains:

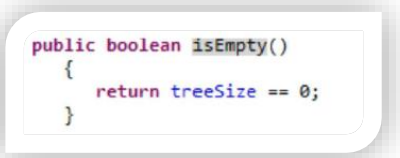
Nos devuelve un booleano indicándonos si el elemento buscado existe en el BStree, para realizar la búsqueda utiliza findnode.

```
vate BSTNode<T> findNode(Object item)
{
   // t is current node in traversal
   BSTNode<T> t = root;
   int orderValue;

   // terminate on on empty subtree
   while(t != null)
   {
      // compare item and the current node value
      orderValue = ((Comparable<T>)item).compareTo(t.nodeValue);

      // if a match occurs, return true; otherwise, go left
      // or go right following search tree order
      if (orderValue == 0)
         return t;
      else if (orderValue < 0)
         t = t.left;
      else
         t = t.right;
   }

   return null;
}
```

Como podemos ver realiza un recorrido de los elementos del árbol y mediante compareto() compara si el elemento que estamos buscando corresponde con el nodo del árbol que estamos actualmente. Si no lo encuentra devuelve nulo.

IsEmpty:

Devuelve true si el BStree está vacío, si contiene algún elemento devolverá false.



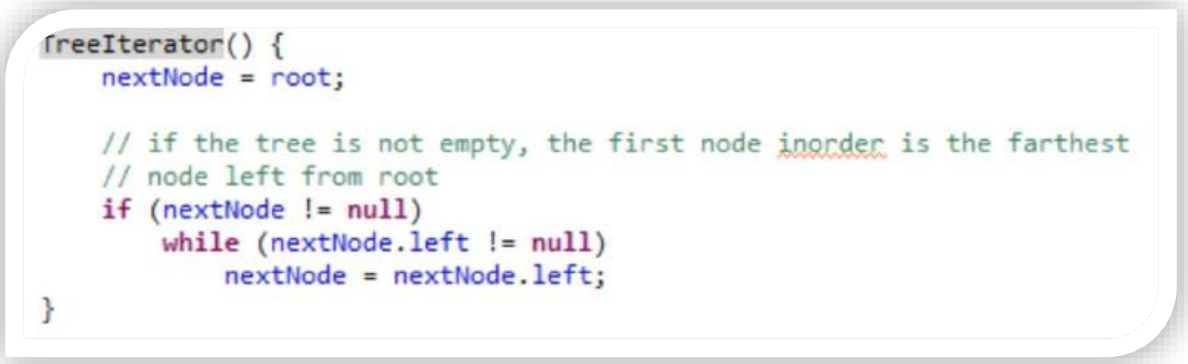
```
public boolean isEmpty()
   {
      return treeSize == 0;
   }
```

TreeIterator:

El iterador será el que nos permita movernos a través de los nodos del árbol. Dispone de los siguientes métodos:

TreeIterator():

Es el constructor de, como podemos ver en el código comprobara que el nodo no es nulo y fijara el recorrido del árbol en el nodo hoja más a la izquierda del árbol.



```
TreeIterator() {
    nextNode = root;

    // if the tree is not empty, the first node inorder is the farthest
    // node left from root
    if (nextNode != null)
        while (nextNode.left != null)
            nextNode = nextNode.left;
}
```

HasNext():

Devuelve un elemento de tipo boolean (true, false). En el caso que el iterador disponga de más elementos que recorrer devolverá true, si está en el elemento final del árbol devolverá false.

Current():

Devuelve el elemento donde se encuentra el iterator.

Next():

Devuelve el próximo elemento del árbol.