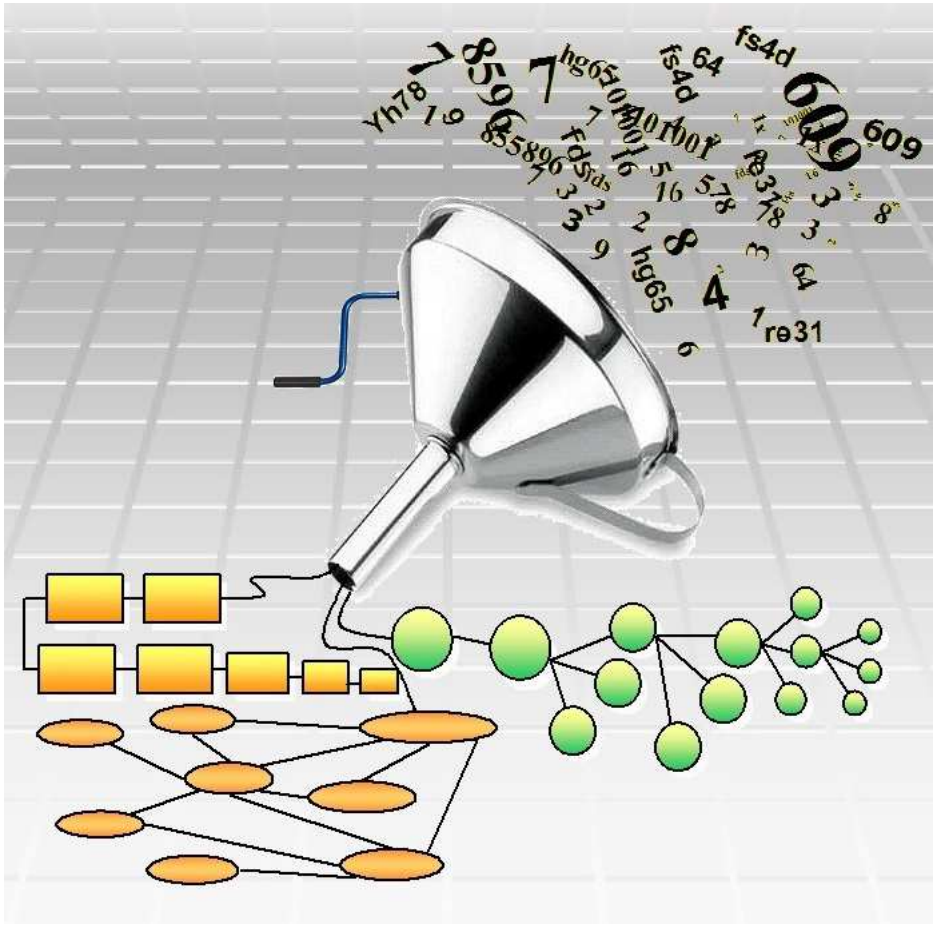
Memoria Practica 02



Autor:	David Subires Parra
Asignatura:	Estructuras de Datos y Algoritmos 1
Grado:	Ingeniería informática
Universidad:	Universidad de Almería
Curso Académico:	2013/2014

Estructura de datos Binary Search Tree (BSTree)

Practica02.Ejercicio01

La clase **BSTree** contiene a su vez dos clases, **BSTNode** y **TreeIterator**:

-**BSTNode**<T>: Clase que implementa el nodo que, combinado con otros nodos, forma el BSTree.
Contiene un atributo "nodeValue" de tipo genérico que será dónde se almacene el valor del nodo, también contiene tres atributos "left, right y parent" del tipo BSTNode<T>, que harán referencia al hijo izquierdo, hijo derecho y padre de dicho nodo.

-**TreeIterator**: Clase que implementa un iterador adaptado al BSTree.
Permite recorrer todos los elementos del ABB empezando por el primer nodo en orden(inorder), el primer nodo hoja empezando por la izquierda situado en el sub-árbol izquierdo mas lejano a la raíz, hasta el último, el primer nodo hoja empezando por la derecha situado en el sub-árbol derecho más lejano de la raíz.
Además, como todo Iterator, nos proporciona los metodos hasNext, current, next y remove. En el caso de next, devuelve el nodeValue, no el BSTNode.
Tiene un único constructor, TreeIterator(), que inicializa el iterador asignando a su atributo nextNode el root del BSTree que lo invoca. Después, como se ha comentado más arriba, se recorre el BSTree hasta que nextNode apunte hacia el nódo más pequeño del BSTree.

Los atributos de la clase BSTree son BSTNode<T> **root**, int **treeSize**, int **modCount**:

- BSTNode<T> **root** → Elemento que contiene la estructura de datos.
- int **treeSize** → Tamaño de BSTNode<T> root
- int **modCount** → Se incrementa cuando root cambia. Lo usa el iterador para comprobar que está en un estado consistente.(el contenido del BSTree es el mismo que el del iterador)

BSTree dispone sólo de un constructor, al que se invoca de la siguiente manera:
BSTree() → La inicialización consiste en asignar null a root, y 0 a modCount y treeSize.

Los métodos más relevantes de **BSTree** son:

- **add**(T item) → Añade item al BSTree.
  Devuelve true si la inserción se realiza correctamente, false en caso de que ya exista el elemento(no añade duplicados), y por lo tanto no lo inserta.
  El proceso que realiza consiste en añadir un nuevo nodo hoja, en el sub-árbol izquierdo o derecho, dependiendo del resultado de comparar el valor de "item" con cada nodo padre de todo sub-arbol, empezando por la raíz del arbol.
  Tiempo medio = O(log n)
  Tiempo peor caso O(n)

- **clear**() → Elimina el contenido del BSTree.
  Consiste en modCont++, treeSize =0 y root = null

- **contains**(Object item) → Comprueba si BSTree contiene el elemento “item”.
  Devuelve true si BSTree contiene “item”, false en caso contrario.
  Consiste en llamar a la funcion findNode() y si devuelve algo distinto de null el BSTree contiene el elemento “item”.

- **isEmpty**() → Devuelve true si el atributo treeSize es igual a 0, false en caso contrairo.

- **Iterator**() → Devuelve un iterador del tipo TreeIterator(Explicado anteriormente).

- **Remove**(Object item) → Elimina el objeto “item”, pasado por parámetro, del BSTree.
  Devuelve true si elimina el elemento, false en caso contrario (Por lo tanto el objeto no está en el BSTree).

Para realizar dicha operación comprueba si el objeto “item” pasado por parámetro está en el BSTree, en caso afirmativo, con éste llama al método privado removeNode(BSTNode<T> (se explicará más adelante).

- **size**() → Devuelve el tamaño(int) del BSTree.

- **toArray**() → Devuelve un objeto del tipo Object[] con el contenido del BSTree.
  Para ello, recorre el BSTree mediante un TreeIterator, y en el orden de éste(inorden) , desde el valor más pequeño hasta el valor más grande, almacena el valor de cada BSTNode en el array Object[] para posteriormente devolverlo.

- **toString**() → Devuelve un String que contiene el valor de todos los nodos del BSTree. Para ello recorre el BSTree mediante un treeIterator y va almacenando cada valor (una vez más en el orden del treeIterator) en el String para posteriormente devolverlo.
  La salida tendría el siguiente formato: [1\n 2\n 3]
- **find**(T) → Busca en el BSTree el nodo con valor(nodeValue) igual a “item” pasado como parámetro. Si lo encuentra, devuelve el valor de dicho nodo, sino, devuelve null.
  Para realizar la operación llama al método privado findNode(se explicará más adelante)
- **clone**() → Devuelve un objeto del tipo Object que contiene el BSTree clonado del objeto BSTree que lo invoca.
  Para ello, instancia nuevo objeto BSTree, lo clona mediante super.clone() y después copia la raiz(root) del BSTree que lo ha invocado al nuevo BSTree, mediante el método privado copyTree() (se explicará más adelante)
- **preorderDisplay**() → Devuelve un String con el contenido del BSTree, en preordern, empezando por el nodeValue de la raiz(root), pasando por todos los hijos izquierdos, y después por los derechos.
- **displayTree**() → Devuelve un String con el contenido del BSTree, añadiendo por cada valor de cada nodo el nivel en el que está. Realiza la operación añadiendo primero al String el hijo más a la derecha, después su padre, después el hijo izquierdo, y así sucesivamente.

Métodos **privados** de **BSTree**:

- **removeNode**(BSTNode) → Elimina el nodo pasado por parámetro del BSTree.
  Para ello, primero se comprueba si el BSTNode pasado por parámetro es igual a null, en ese caso se devuelve null. En caso negativo, se comprueba si uno de los dos hijos del nodo a elimiar es igual a null, en ese caso, se elimina el nodo en cuestión y su lugar lo ocupará el hijo no nulo.
  En caso de que ninguno de sus hijos sea igual a null, el nodo eliminado es reemplazado por el nodo hoja más a la izquierda del sub-arbol derecho del nodo reemplazado.

- **findNode**(Object item) → Devuelve el BSTNode con el nodeValue "item" pasado por parámetro si lo encuentra, null en caso contrario.
  Para ello, empieza a recorrer el BSTree desde el root, comparando los valores de cada nodo con "item", si el valor de item es menor, pasa a comprobar item con el valor del hijo izquierdo, si item es mayor, comprueba el hijo derecho. Si realizando el recorrido da con el valor buscado, lo devuelve, sino devuelve null.
  Tiempo medio = O(log n)
  Tiempo peor caso O(n)

- **copyTree**(BSTNode<T> t) → Devuelve un BSTNode con el mismo contenido/estructura del BSTNode pasado por parámetro.
  Para ello, se llama a sí mismo recursivamente, primero con el hijo izquierdo de cada nodo, empezando por root, después el hijo derecho y finaliza cada llamada recursiva con con el valor de t.