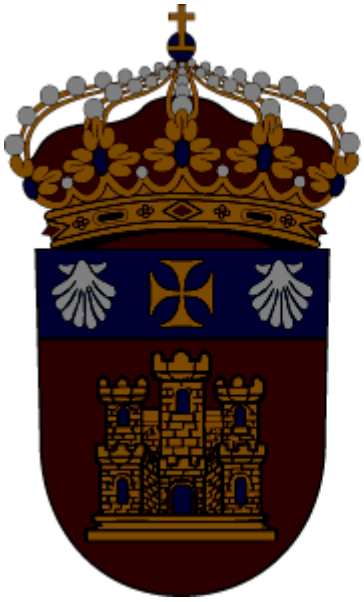
Sergio Barbero Báscones

Estructuras de Datos

Práctica 11



UNIVERSIDAD DE BURGOS

Contenido

1. INTRODUCCION

Se espera que el alumno se familiarice con la implementación interna de algunas de las estructuras de datos que se incluyen en el *Java Collections Framework*, así como las posibilidades que existen para extender la jerarquía con nuevas implementaciones.

En el caso de la práctica actual se trabajará sobre la implementación interna de un **Set**, construido como un **Árbol Binario de Búsqueda**, creando una implementación inmutable de Set para incluirse dentro de las librerías de Java. El objetivo de la práctica es que el alumno se familiarice con los algoritmos explicados en teoría para la inserción ordenada de elementos y el cálculo de algunas propiedades del árbol.

Al igual que en anteriores prácticas se trabajará con la Genericidad de Java, que permite definir los tipos de datos a almacenar en una colección. Es una característica muy importante que mantienen todos los componentes del *Java Collections Framework*.

2. ESTRUCTURA GENERAL

Debemos generar una clase padre y dos clases hijas.

La clase padre representará la estructura interna del árbol binario de búsqueda.

Una de las clases hijas representará a cada uno de los nodos del árbol

Otra de las clases hijas representará al iterador del árbol binario de búsqueda.

3. DESCRIPCIÓN DE LOS MÉTODOS

Clase ArbolBB<T>

- `Boolean comparable()`: Muestra si el árbol dispone de un comparador (no es comparable) devolviendo false, o por el contrario el comparador es null (es comparable) devolviendo true.

  Complejidad algorítmica: O(1)

  Únicamente existe la comprobación de si un atributo es nulo o no.

- `Iterator<T> iterator()` Nos devuelve una instancia del iterador del árbol binario de búsqueda.

Complejidad algorítmica: O(1)

Únicamente devuelve una instancia del iterador.

- `Int size():` Nos devuelve el tamaño del árbol (numero de elementos del árbol).

  Complejidad algorítmica: O(1)

  Número de elementos del set.

- `Int altura(T elemento):` Nos devuelve la altura de un elemento desde el ultimo nodo hoja relacionado con el.

  Complejidad algorítmica: O(2log n)

  El método itera el árbol en función del elemento proporcionado (escogiendo ir por el lado derecho si el elemento es superior a la raíz o por el izquierdo en caso contrario). Una vez encontrado el elemento itera el árbol hasta la hoja más profunda comprobando si alguno de sus hijos no es nulo.

- `ArrayList<T> inOrden()` Nos devuelve una lista con el recorrido inOrden del árbol, el método esta implementado de forma iterativa.
  Mientras el elemento en el que estoy no sea nulo o la pila no este vacía repito esta operación: Si el elemento en el cual estoy no es nulo introduzco dicho elemento en la pila y establezco actual al hijo izquierdo, en caso contrario habremos llegado al nodo hoja por lo que sacaré un elemento de la pila y lo añadiré al recorrido inOrden y establezco actual al hijo derecho.

  Complejidad algorítmica: O(n)

  Recorro todo el árbol.

- `Int profundidad()` Nos devuelve la profundidad en la que se encuentra un elemento con respecto al nodo raíz

  Complejidad algorítmica: O(log n)

  Recorro el árbol según los criterios de que mi elemento sea mayor o menor que la raíz (log n), a medida que voy haciendo eso incremento un contador, hasta encontrar el elemento.

Clase Nodo<T>

Únicamente contiene al constructor, se le pasa un elemento y lo introduce en el nodo.

Clase IteradorArbolBB

- `Boolean hasNext():` Muestra si el árbol dispone de un elemento siguiente

  Complejidad algorítmica: O(1)

  Comprueba el elemento siguiente.

- `T next()` Nos devuelve el elemento siguiente

  Complejidad algorítmica: O(1)

  Devuelve el elemento siguiente en la iteración.