PRACTIA 2 EJERCICIO 1: ARBOLES BSTREE

Juan Miguel Herrada Acosta

Juan Miguel Herrada Acosta
Grado en ingeniería informática. Estructura de datos y algoritmos I

Se ha proporcionado todo lo relativo a la implementación del BSTree, por lo que se pide que se EXPLIQUE en un documento (memoria) toda la implementación suministrada, prestando especial atención a funciones principales de dicha estructura de datos como: add, remove, clear, contains, isEmpty y al uso propuesto del iterador asociado. Razone y justifique adecuadamente cada una de las ventajas e inconvenientes del uso de esta estructura comparada con el uso de estructuras lineales.

Métodos:

- public boolean add (T item): añade un elemento al árbol si el elemento ya se encuentra dentro del árbol devolverá falso y si lo pudo añadir sin problemas devolverá verdadero.
  Internamente inicializamos un nodo igual a la raíz, un padre igual a null y un hijo declarado. Seguidamente Comprobamos si el raíz es distinto de null comprobara el valor de orden comparado primero con la raíz y seguidamente dependiendo si el valor es menor que cero con el hijo izquierdo o el hijo derecho si en algún caso el valor es igual a cero devolvería false ya que el valor ya se encuentra en alguno de los nodos del árbol, en caso contrario guardaríamos un valor de orden para su posterior inserción en el árbol.

  Inicializamos el nodo a añadir y comprobamos si el padre es igual a null, es decir, el árbol está vacío y el nodo a añadir seria la raíz. Si la anterior opción no se cumple pasaríamos a comprobar el valor de orden para identificar en que rama se añadirá y si será el hijo izquierdo o derecho.

  Para Finalizar incrementamos en uno el tamaño del árbol, el contador para saber si el árbol es consistente y devolvemos verdadero.

- public boolean remove (T item): elimina un elemento del árbol si el árbol contiene dicho elemento lo elimina y devuelve verdadero, si no lo contiene devuelve falso.
  Internamente inicializamos un nodo al valor de la búsqueda del elemento dentro del árbol, este valor se utilizara para comprobar si el nodo se encuentra o no dentro del árbol si el valor es igual a null devolveremos false sino pasaremos el nodo inicializado a un método privado llamado removeNode el cual eliminara ese nodo recorriendo el árbol hasta encontrar y recalibrándolo si fuese necesario.
  En caso de contener dicho nodo en nuestro árbol reduciríamos en uno el valor del árbol, aumentaríamos el contador de consistencia y devolveríamos verdadero.

- public void clear (): elimina todos los elementos del árbol dejándolo vacío.
  Internamente incrementa en uno el valor del contador de consistencia, iguala a cero el tamaño del árbol e iguala el nodo raíz a null.

- public boolean contains (T item): devuelve verdadero si nuestro árbol contiene dicho elemento y falso en caso contrario.
  Internamente hace una llamada al método find (este método devuelve el nodo que contiene ese elemento o null si el elemento no se encuentra en el árbol), si el valor devuelto es igual a null el método devolverá falso y en caso contrario devolvería verdadero.

- public boolean isEmpty (): comprueba si el árbol está vacío o no.
  Internamente comprueba el tamaño del árbol es igual a cero, en tal caso devolveríamos verdadero y en caso contrario falso.

- public Iterator<T> iterator (): devuelve un iterador genérico de tipo T.
  Internamente devuelve la creación de un TreeIterator () este método es el constructor de la clase iterator interna de BSTree.
  Esta clase es necesario porque la necesidad de un iterador para recorrer la colección.
  TreeIterator () internamente igualamos el siguiente nodo a raíz y comprobamos si el siguiente nodo es distinto de null, cuando sea distinto de null entraremos a comprobar que mientras el hijo izquierdo del siguiente nodo sea distinto de null igualaremos el siguiente nodo a su hijo izquierdo.

Ventajas:

- El coste de recorrer BSTree es $O(\log n)$ comparado con las lineales que sería de $O(n)$ o $O(n * \log n)$.
- La inserción de valores ya tiene un orden, es decir, no hace falta ordenar la colección.

Inconvenientes:

- El árbol puede degenerarse si la inserción de los datos esta ordenada y en el peor de los casos el coste sería $O(n)$.