IMPLEMENTACIÓN ÁRBOL-B

Miguel Ascanio Gómez

Esta implementación se ha realizado siguiendo la especificación en el Cormen et. Al. con ligeras modificaciones, para estar también acorde a lo especificado por java.util.ShortedMap:

1. Se implementa como un mapa <Clave, Valor>, en el que la Clave debe ser comparable con ella misma (extiende a comparable de Clave)
2. El método de Inserción devuelve null al insertar un par (clave, valor) si no existía una entrada en el TAD con la clave que se inserta; o si la clave ya existía, se devuelve el valor ya existente, y posteriormente se cambia por el que se está insertando.
3. El método de Búsqueda(clave) devuelve null si no existe la clave insertada; o el valor asociado a dicha clave si existe.
4. El método de borrado(clave) devuelve null si la clave no existía, o el valor asociado a dicha clave si estaba contenida en el árbol.
5. En esta implementación no se permiten repeticiones, si bien si se permite insertar una clave existente para sobrescribir el valor asociado ya existente.
6. Se implementan, además, una función para devolver el inorden del árbol (como arrayList), y dos funciones para obtener la clave más pequeña y la más grandes del árbol.
7. Se permite además definir una función, tipo función hash, para establecer una relación valor->clave. Esta función, definida dentro de una clase abstracta, se pasa opcionalmente como parámetro a la constructora.

Pruebas

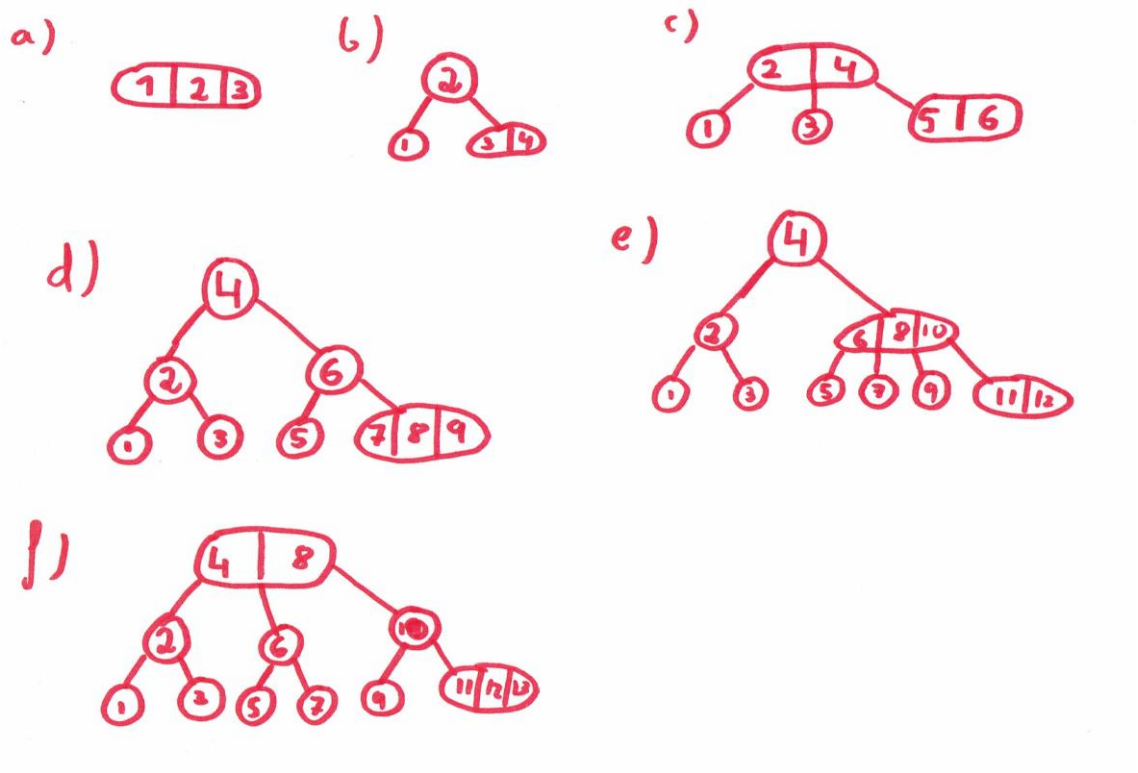
Para probar el funcionamiento correcto de la implementación, se ha desarrollado la siguiente función para validarlo:

1. Se realiza una inserción de elementos aleatorios, y para cada elemento se comprueba que el inorden del árbol está ordenado. Los mismos elementos se introducen en un array auxiliar.
2. Se comprueba que el Árbol y el array contienen los mismos elementos
3. Se realiza una permutación del array, y se van borrando los elementos del Árbol. Para cada borrado se comprueba que se borra el elemento correcto (la clave tiene que ser la misma que el valor, ya que así se ha insertado), y para cada borrado se comprueba que el inorden sigue ordenado.
4. Se realiza una comprobación de PutAll, probanto que el árbol contiene todos los elementos de un Árbol nuevo de la clase TreeMap (implementación de un rojinegro)

Casos de prueba concretos

Inserción

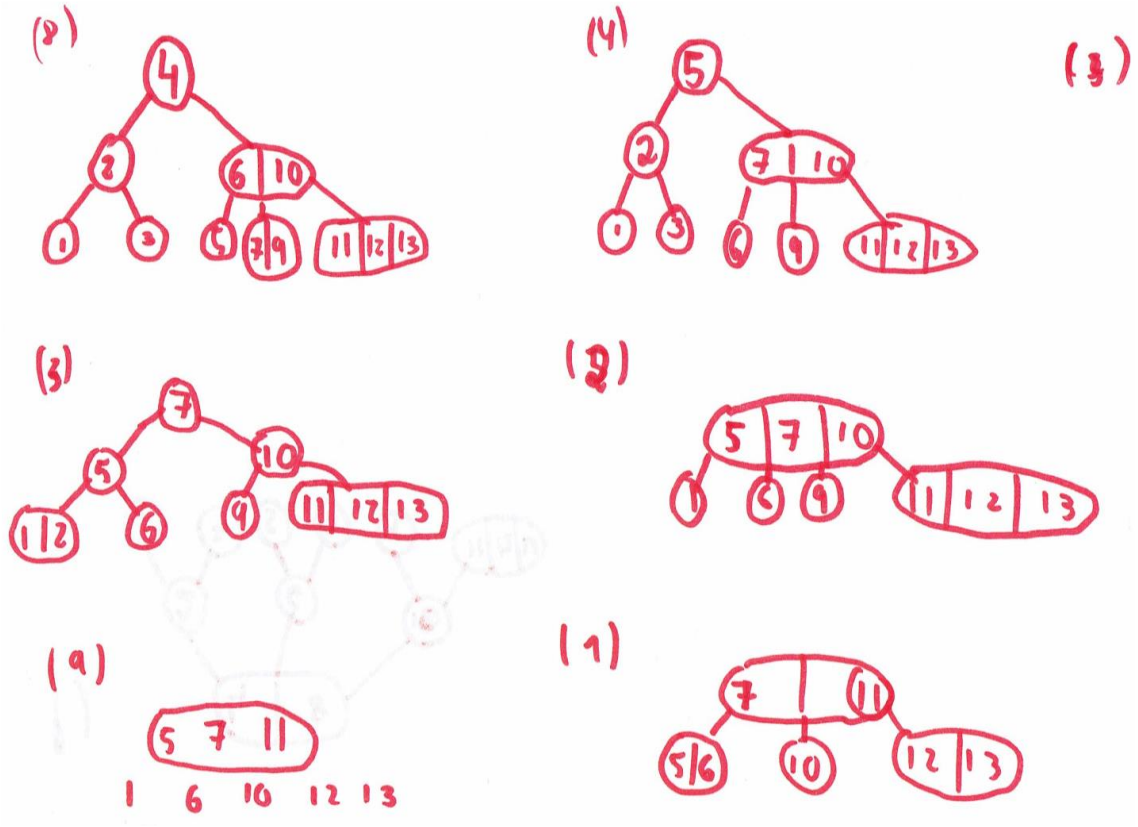
A continuación se muestran diversas fases de la inserción de la secuencia 1-13 en un árbol de grado 2:



Se insertan los números del 1 al 3 en las entradas del nodo raíz

a) Insertar una enrtada más en la ráiz violaría el invariante nClaves < 2t-1, por lo tanto hacemos un Split del nodo raíz, dejando como nueva raíz el nodo 2 y teniendo como hijos el 1 y el 3. Acto seguido insertamos el 4 sin problemas.
b) Tras insertar el 5 y al intentar insertar el 6 en el nodo más a la derecha, éste quedaría con 4 entradas lo que no está permitido: subimos la entrada 4 a la raíz, y le ponemos como hijo izquierdo el 3, el derecho sigue siendo el 5. A continuación insertamos el 6 sin problemas.
c) Tras insertar el 7 sin problemas, y el 8 catalpuntando un nodo más a la raíz (llegando a tener la raíz (2,4,6)), intentamos inertar el 8: la raíz está llena, y en la inserción estamos garantizando que siempre dejamos como máximo 2t-2 entradas en la bajada, lo que se hace en este punto es poner el 4 como raíz, con hijos 2 y 6- Posteriormente se inserta el 9
d) Se inserta el 10, 11 y 12, el hijo derecho de la raíz va creciendo al estar el hijo inferior derecho catalpuntando nodos hacia arriba.
e) Insertamos el 13: al bajar por la raíz al hijo derecho, nos damos cuenta que el nodo está lleno, por lo tanto hay que subir algo a la raíz. Subimos la entrada 8 a la drerecha de la raíz, ponemos la entrada que estaba a la izquierda del 8 (el 6) como hijo a la izquierda del 8, manteniendo el 6 los dos hijos que tenía anteriormente. Esto deja al hijo derecho de la raíz (colgando ahora del 8) con una entrada 10, con hijo izquierdo el 9 e hijo derecho el 11,12. Ahora ya podemos insertar el 13 a la derecha del 12.

Borrado

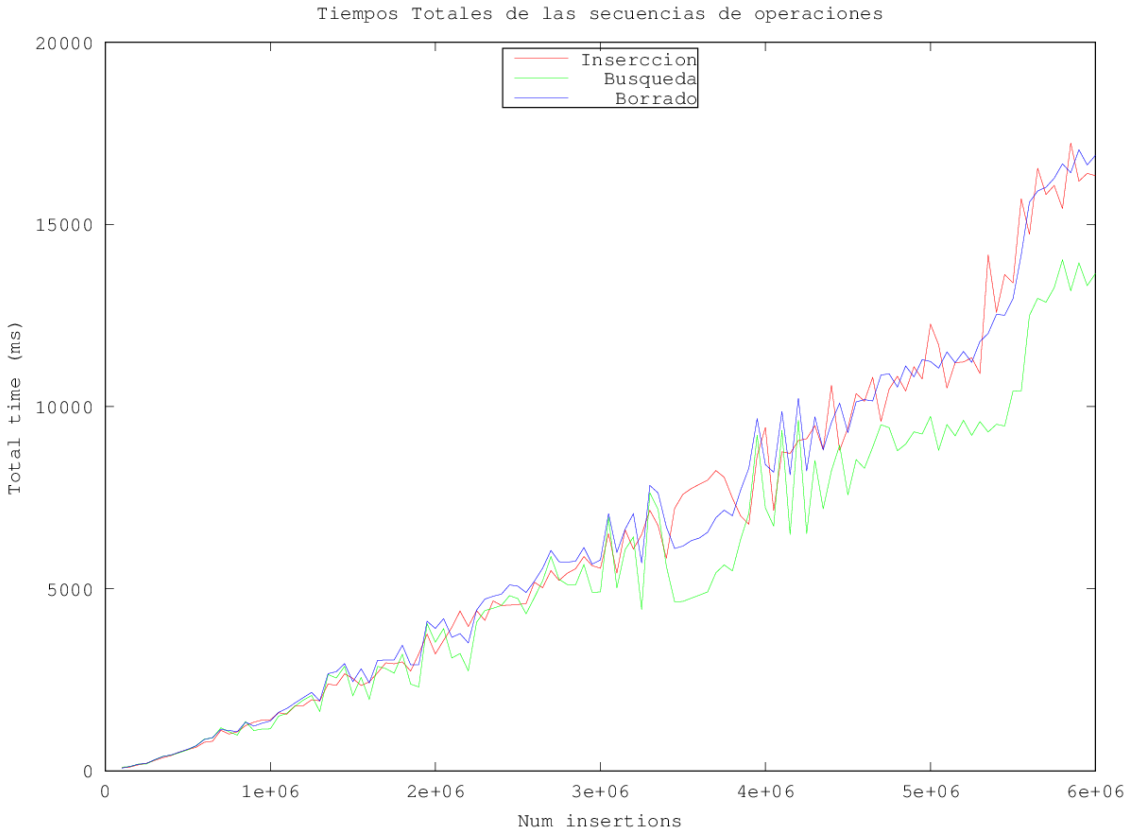


- (8) Al borrar el 8 fusionamos el 6 y el 10 en un nuevo nodo, así como el hijo más grande del 6 y el menor del 10 los fusionamos en el (7,9)
- (4) Al borrar el 4, la raíz quedaría vacía, por lo que tenemos que subir algo de uno de los hijos, del izquierdo es imposible porque quedaría vacío, así que lo hacemos del derecho: subimos la entrada más pequeña de todo el sub-árbol derecho del 4 (el 5); en este proceso el 6 baja como raíz, el 7 asciende para colocarse en el lugar del 6 y el 5 se coloca como la nueva raíz.
- (3) Bajo de la raíz (se permite a la raíz tener menos de t-1 entradas, en caso de quedar a 0 entradas se pone el hijo (que sería único) como raíz); ahora no podría bajar ya que el nodo actual solo tiene una entrada (el 2), lo que se hace en este punto es lo siguiente: como el hermano derecho todavía puede perder una entrada, bajo el 5 de la raíz, subo el 7 para ser la nueva raíz, y pongo el 6 como hijo derecho del 5 (nótese que el 6 sigue a la derecha del 5 y a la izquierda del 7), ahora ya se puede bajar del nodo (2,5) a borrar el 3, para lo cual el 2 acaba bajando a una hoja (ya que al borrar el 3 una hoja quedaría vacía)
- (2) Como antes, bajo por la raíz hasta el 5, el problema es que no puedo actuar como antes, pues el hermano derecho no puede perder una entrada: en este caso, bajo una entrada del padre (bajo el 7 y la raíz se queda vacía) y fusiono con el hermano derecho. Acto seguido borro el 2 de la hoja sin problemas. En la vuelta de la llamada recursiva, al llegar a la raíz (vacía, con un único hijo (5,7,10)), cambio la raíz del árbol a dicho nodo.
- (9) Al borrar el 9 se quedaría una hoja vacía, como el hermano derecho puede perder una entrada, bajo la entrada del padre a la derecha, y la sustituyo por la menor del hermano.
- (1) Al borrar el 1 se quedaría la hoja vacía, como el hermano derecho no puede perder una entrada, lo que se hace es fusionar los dos hermanos tomando una entrada del padre.

Como se puede observar, la idea del borrado es siempre bajar, garantizando que el nodo tiene al menos t entradas (por si se diera el caso que en la llamada recursiva se eliminara una entrada de dicho nodo, quedaría con t-1 entradas lo cual es correcto). Esto es análogo a la inserción: siempre se asegura que el nodo del que se baja no está lleno, por si la llamada recursiva subiera una entrada a dicho nodo.

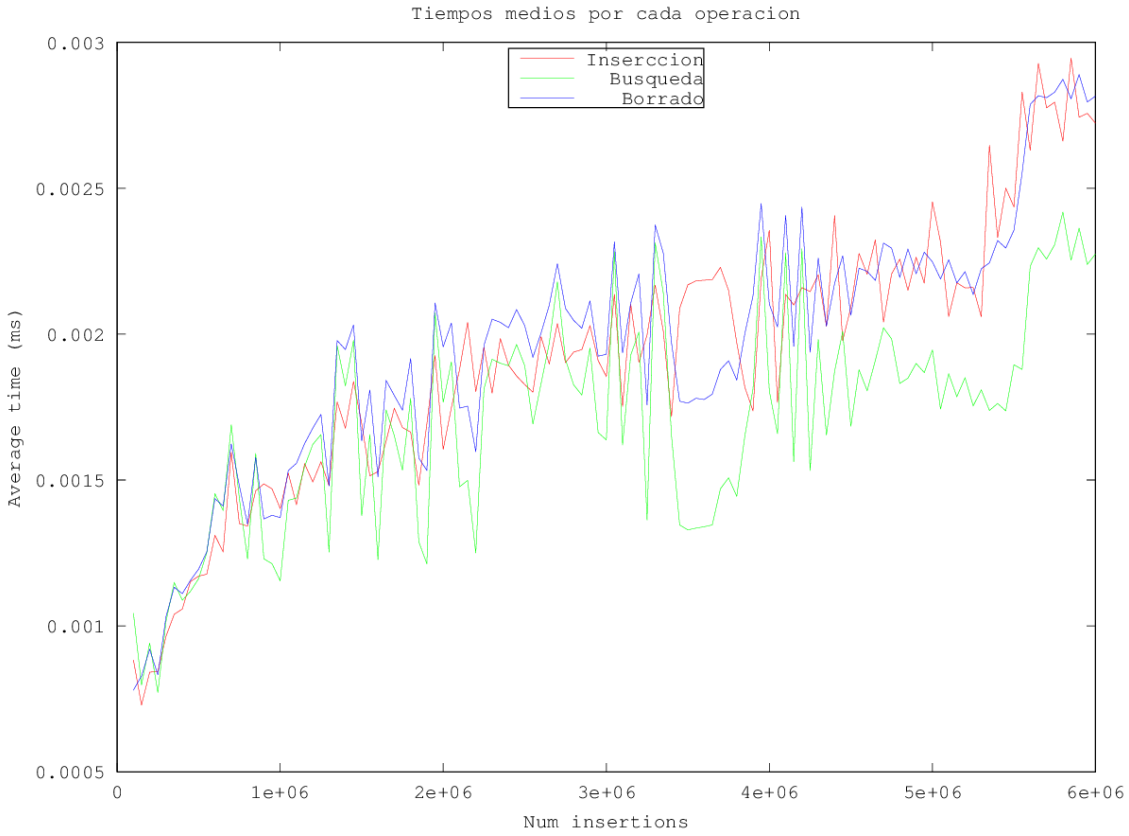
Rendimiento

Se han realizado pruebas de rendimiento sobre el árbol. Las mismas constan de entre 100k y 6M de elementos, a intervalos de 50k; en cada una se realiza la inserción, búsqueda y borrado, en dicho orden, de los elementos pertinentes (generados aleatoriamente), repitiendo cada conjunto de operaciones 3 veces con elementos diferentes, guardando el tiempo medio de las tres.



La gráfica tiene forma de curva nlogn, lo cual tiene sentido: se realizan n operaciones de coste logn, siendo n el número de elementos en el árbol.

En la próxima curva, se presentan los valores de la anterior gráfica divididos entre el número de elementos, lo que nos da el tiempo medio por operación:



Se intuye que se aproxima a una curva logarítmica, que es precisamente lo esperado.