EDA (E.T.S. de Ingeniería Informática)
Curso 2016-2017
Práctica 4. Árboles Binarios de Búsqueda

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

1. Objetivos

- Aprender a evaluar la forma de un árbol binario de búsqueda (ABB) y a construir un árbol binario de búsqueda equilibrado.
- Utilizar eficientemente un árbol binario de búsqueda equilibrado para implementar un *Editor Predictivo*.

2. Contexto y trabajo previo

Para que aproveches al máximo la sesión de prácticas, antes debes realizar una lectura comprensiva de este boletín y familiarizarte con el código de las clases que se te proporcionan a través de PoliformaT; en concreto observa qué atributos tienen las clases y la funcionalidad de cada una de ellas. El árbol binario de búsqueda (ABB) es la estructura de datos no lineal más básica. Los ABB se definen de manera recursiva como un conjunto finito de nodos que puede ser vacío (caso base), o un nodo raíz con un par de ABB como hijos izquierdo y derecho (caso general) con la propiedad de que cualquier valor en el sub-árbol izquierdo ha de ser menor que el de la raíz y cualquier valor en el sub-árbol derecho ha de ser mayor que el de la raíz.

Conceptos importantes:

- Un árbol degenerado es aquel para el que cualquier nodo interno solamente tiene un hijo. Esto es, el árbol tiene forma de lista.
- Un árbol lleno es aquel cuyos nodos internos tienen siempre dos hijos.
- Los costes para la búsqueda y la inserción dependen de la altura del árbol y oscilan entre $O(log_2(n))$ para un árbol completo y $O(n)$ para el caso del árbol degenerado.

3. Actividades en el laboratorio

3.1. Construcción equilibrada de un ABB

Las clases `ABB` y `NodoABB` contienen el código necesario para la implementación de un ABB. Estas clases deberán añadirse al paquete *librerias/estructurasDeDatos/jerarquicos*.

La forma que tiene un ABB depende del orden de inserción de los elementos en él. Si insertamos los elementos de manera inteligente podemos controlar la forma que tendrá el árbol.

- Implementa el método `protected NodoABB<E> construirEquilibrado(E[] v, int ini, int fin)` para que, al invocarlo, construya el ABB dejándolo equilibrado. Este método recibe un array que contiene todos los valores ordenados a insertar en el árbol.

- Completa el constructor `public ABB(E[] v)`, que crea un ABB equilibrado con los elementos contenidos en el array. Para ello se debe utilizar el método anterior, *construirEquilibrado*, teniendo en cuenta que el array v no tiene porqué estar ordenado.

- Completa el método `public void reconstruirEquilibrado()`, que reconstruye el ABB, con los mismos datos que tiene actualmente, de forma que quede equilibrado. Para ello, se deberá usar también el método *construirEquilibrado*; el array con los elementos ordenados que necesita este método puede obtenerse fácilmente a partir de un recorrido in-orden del árbol.

Para implementar el método *construirEquilibrado* procederemos de la siguiente manera: Insertaremos primero la mediana (valor que ocupa la posición central en el array ordenado). Luego insertaremos las medianas de las dos mitades del array, y repetiremos este proceso para cada subarray hasta que el tamaño del subarray sea 0.

Algorithm 1: Método construirEquilibrado

Input: Array con los elementos ordenados del árbol
Input: Índices inferior y superior del array
Output: Árbol equilibrado
if *tamaño sub-array* > 0 **then**
    insertar en el árbol el elemento que ocupa la posición central del sub-array;
    construirEquilibrado para el sub-array izquierdo;
    construirEquilibrado para el sub-array derecho;

Para conseguir una implementación más eficiente es conveniente no usar el método *insertar* de *ABB* y construir directamente los nodos del árbol. Discute el coste que tendrá la operación completa de regenerar el árbol y bajo que condiciones puede ser interesante realizar esta operación.

3.2. Validación de la clase *ABB*

Utiliza la clase `TestABB`, que deberás incluir en el paquete *librerias/estructurasDeDatos/jerarquicos*, para verificar que la reconstrucción equilibrada del *ABB* que has implementado funciona correctamente. Pruébalo con un árbol equilibrado, con uno degenerado y con uno, al menos, generado aleatoriamente para asegurarte.

El test te permitirá también observar el funcionamiento de los métodos de inserción y borrado de elementos en un ABB, así como comprobar el resultado de los recorridos que pueden realizarse sobre él (por niveles, in-orden, pre-orden y post-orden).

3.3. Implementación del Editor Predictivo

Un Editor Predictivo es un editor de texto que conforme un usuario va escribiendo una palabra le ofrece una serie de sugerencias para completarla, sin que tenga necesariamente que terminar de escribirla; esta es, por ejemplo, una aplicación típica en los dispositivos móviles para ayudar a la redacción de pequeños mensajes de texto (en los teléfonos móviles recibe el nombre de T9 o xT9).

Figura 1: Ejemplo de Editor Predictivo.

Para completar el funcionamiento del Editor Predictivo se deberán seguir los siguientes pasos:

- Añadir la clase `EditorPredictivo` al paquete *aplicaciones/editorPredictivo*. Así mismo, el alumno deberá copiar el fichero `castellano.txt`, disponible también en PoliformaT, en el directorio asociado al paquete *aplicaciones/editorPredictivo*; este fichero contiene un gran subconjunto de palabras del Castellano ordenadas alfabéticamente (los datos con los que, por defecto, se construirá un Editor Predictivo).

- Completar el método constructor de la clase `EditorPredictivo` que, dado un fichero de palabras ordenado ascendentemente, crea un `ABB` equilibrado de `String`. Ya se dispone del código para cargar las palabras del fichero en un array ordenado llamado *palabras*. Falta hacer uso del método *construirEquilibrado* para insertar las palabras del array, obteniendo así un ABB equilibrado.

- Completar el método *recuperarSucesores* de la clase `EditorPredictivo`, cuyo perfil es el siguiente:

  ```
  public ListaConPI<String> recuperarSucesores(String prefijo, int n)
  ```

  Este método devuelve una Lista Con Punto de Interés con los `n` primeros sucesores de un `prefijo` dado; recuérdese que esta lista debe incluir a `prefijo` como primer elemento de la lista siempre y cuando este ya figure en el ABB (es decir, que el prefijo sea ya

una palabra completa). Para implementar este método se puede hacer uso del método `sucesor` de la clase `ABB` como sigue:

1. Buscar el prefijo en el `ABB`, por si este fuera ya una palabra completa.
2. Recuperar del `ABB` los siguientes sucesores del prefijo hasta encontrar uno que ya no comience por dicho prefijo.

3.4. Prueba del Editor Predictivo

La clase `TestEditorPredictivo` es un programa que usa la clase `EditorPredictivo` para, en modo gráfico, permitir al usuario la edición de mensajes de texto, ofreciendo sugerencias para completar las palabras en curso. El alumno deberá añadir esta clase al paquete *aplicaciones/editorPredictivo*. Además de probar con el ejemplo de la Figura 1, a continuación se muestran las sugerencias que deben aparecer para una serie de prefijos de prueba:

Prefijo	**Sugerencias**			
catar	catar	catarata	catarral	catarro
	catarroso	catarsis		
mar	mar	mara	marabunta	maraca
	maraco	maracucho	maracuyá	maraquear
	maraquero	marar	marasmo	maratoniano
	maratón	maravilla	maravillar	maravillosamente
	maravilloso	maraña	marañero	
tene	tenebrosidad	tenebroso	tenedor	teneduría
	tenencia	tener	teneraje	tenerife
	tenería			
criti	criticable	criticador	criticar	criticastro
	criticidad	criticón	critiquizar	

Cuadro 1: Ejemplos de prueba para validar la práctica.