Ejercicios Tema 11

Ejercicios Adaptados de Apuntes y Exámenes de EDA
Germán Moltó Martínez
gmolto@dsic.upv.es
Estructuras de Datos y Algoritmos
Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

Versión Iterativa de recuperar en un ABB

- Implementa una versión del método *recuperar* iterativa con la siguiente especificación:

```
protected NodoABB<E> recuperar (E clave, NodoABB<E> n) throws
ElementoNoEncontrado;
```


Versión iterativa de insertarConDuplicados

- Diseñar el método insertarConDuplicados de la clase ABB, siguiendo una estrategia iterativa, como se indica en el siguiente perfil:

```
protected NodoABB<E> insertarConDuplicados(E clave, NodoABB<E> n);
```

- El método debe realizar la inserción en el ABB permitiendo la existencia de elementos duplicados.

Versiones Alternativas

- Diseña la versión iterativa del método *recuperarMax* que busca el mayor elemento de un ABB:

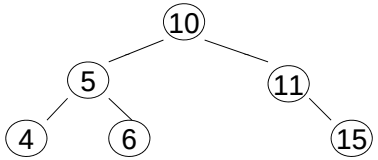
```
protected NodoABB<E> recuperarMax(NodoABB<E>
n);
```

- Diseña la versión recursiva del método *recuperarMin* que busca el menor elemento de un ABB:

```
protected NodoABB<E> recuperarMin(NodoABB<E>
n);
```

Número de Hojas de ABB

- Se desea añadir a la clase ABB un nuevo método para obtener el número de Hojas de un ABB.
- Diseñese el método en la clase ABB como invocación a su homónimo recursivo en la misma clase.



- ¿Cuál es el orden en el que se producen las llamadas recursivas si queremos calcular el número de hojas del árbol?
- ¿Cuál es el orden en el que terminan de ejecutarse las llamadas recursivas?

Borrar Hojas de ABB

- Diseña un método en la clase ABB para que borre todas sus hojas.

Esfuerzo Medio de Comparación Óptimo

- Diseña el método eMCOptimo en la clase ABB que devuelve el eMC que tendría un árbol perfectamente equilibrado con el mismo tamaño (que el del árbol sobre el que se invoca la operación).

Mostrar los Nodos de un Cierto Nivel

- Diseña un método en la clase ABB que permita mostrar el contenido de los Nodos que pertenecen a un determinado nivel (de profundidad).
- public String **toStringNivel**(int k)

Suma de Elementos Mayores o Igual

- Dado un ABB de Integer, se desea sumar los valores de aquellos Nodos que sean mayores o iguales que un valor entero dado.
- Incorporamos a una subclase de ABB (ABBInteger) el siguiente método:

  public int **sumarMayorOIgual**(int x)
- Usando la definición recursiva de árbol, la suma de elementos >= x se define como:
  - La suma de su raiz, si es mayor o igual que x,
  - más la suma de los elementos >= x de su subárbol izquierdo,
  - más la suma de los elementos mayores o iguales que x de su subárbol derecho.
- Se pide: Implementar dicho método y la cabecera de la clase ABBInteger.

Construcción ABB Equilibrado

- Sea un array v de v.length Objetos distintos y ordenados ascendentemente. Para construir eficientemente un Árbol Binario de Búsqueda **equilibrado** con los mismos elementos que v, se decide añadir el siguiente constructor a la clase ABB:

```
public ABB(E v[]){
    raiz = seConstruye(v,0,v.length-1);
}
```
- Dado un ABB vacío, insertarle en secuencia los N elementos de v origina un ABB completamente degenerado.

1. ¿Cómo es posible construir un ABB equilibrado a partir del array v?
2. Escribir el método Java seConstruye.
3. Analizar la complejidad temporal del método seConstruye (talla, instancias significativas, ecuaciones de recurrencia, cotas).

Cálculo del Sucesor

- Implementar un método **iterativo** que obtenga el sucesor de un NodoABB en un ABB:

  protected NodoABB<E> sucesor(E clave, NodoABB<E> n) throws ElementoNoEncontrado;
- Se recuerda que el sucesor de un nodo se puede calcular de la siguiente manera:
  - Si tiene subárbol derecho, es el mínimo elemento de este subárbol.
  - Sino, es el ascendiente por la derecha más cercano.

toString en ABBColaPrioridad

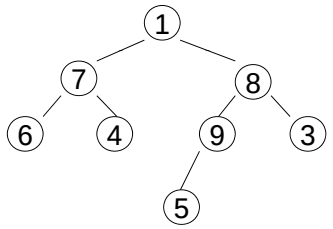
- Implementa el método toString en la clase ABBColaPrioridad para que muestre los elementos ordenados de manera creciente en base a su prioridad.
- Calcular el coste temporal asintótico del método en toda su extensión:
  - Talla del problema en función de los argumentos de entrada.
  - Instancias significativas.
  - Relaciones de recurrencia.
  - Cotas de complejidad temporal asintótica.

Cambia Signo Árbol Binario de Búsqueda

- Dentro de una clase que extiende *ABB* llamada *ABBInteger* que representa árboles binarios de búsqueda que sólo contienen objetos *Integer*, se desea diseñar un método que cambie el signo de todos los datos del árbol.
- Se pide:
  - Diseñar la cabecera de la clase.
  - Diseñar el método vía o lanzadera *cambiaSigno*
  - Diseñar el método interno cambiaSigno
- Nótese que la propiedad de ordenación del ABB se debe seguir manteniendo.

Padre de un Nodo en un ABB

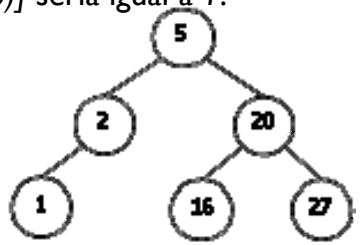
- Diséñese un método iterativo que obtenga el Padre de un nodo de un Árbol Binario de Búsqueda. Si el nodo especificado no existe, se devolverá null.



- ab.padre(new Integer(8)) devuelve un Integer(1)
- ab.padre(new Integer(1)) devuelve null

Fuera de Intervalo (1/2)

- Sea un par de *objetos x,y* que definen un intervalo no vacío *[x,y]*; se quiere definir una nueva operación sobre un *Árbol Binario de Búsqueda* que obtenga el número de sus datos que están **fuera** de dicho intervalo.
- Por ejemplo, en el siguiente *ABB* de *Integer* el resultado de dicha operación sobre el rango *[new Integer(1), new Integer(20)]* sería igual a *1*.



Fuera de Intervalo (2/2)

- Se pide:

a) Diseñar en la clase *ABB* un método lanzadera *fueraDeRango* que dados un par de *objetos x,y* obtiene el número de datos que están **fuera** del intervalo *[x,y]* en el *ABB* sobre el que se aplique.

b) Diseñar en la clase *ABB* el método interno *fueraDeRango* que dados un par de *objetos x,y* y un *NodoABB<E> actual* obtiene, con el mejor coste posible, el número de datos que están fuera del intervalo *[x,y]* en el *NodoABB<E>* sobre el que se aplique.

- **NOTA:** Los datos del ABB no tienen por qué ser *Integer* como en el ejemplo

Coste Temporal de Contar Mayores

- Analiza la complejidad temporal asintótica del siguiente método recursivo:

```
//Obtiene el número de datos del nodo actual de un ABB mayores que x
protected  int contarMayoresQue(E x, NodoABB<E> actual) {
      int res = 0;
       if ( actual != null ){
           if  ( actual.dato.compareTo(x) > 0 ) res += 1;
           res += contarMayoresQue(x, actual.izq);
           res += contarMayoresQue(x, actual.der);
       }
       return res;
}
```

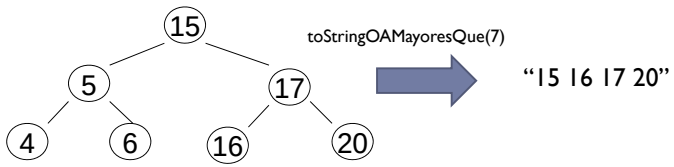
- Se pide: Talla, instancias significativas, ecuaciones de recurrencia y cotas.

Solución Eficiente a Contar Mayores

- El método de la transparencia anterior para contar el número de datos mayores que uno en un ABB puede resolverse de manera más eficiente.
- Utiliza la propiedad de ordenación del ABB, asumiendo que no existen elementos duplicados para resolver el mismo problema de manera más eficiente.

toString Ordenado Ascendentemente

- Se desea añadir un nuevo método a la clase ABB para que obtenga un String en el que aparezcan ordenados ascendentemente aquellos datos del ABB que sean mayores que uno dado.
  - Si no hay ninguno, el método lo advertirá lanzando la excepción ElementoNoEncontrado.
  - El método se llamará toStringOAMayoresQue.

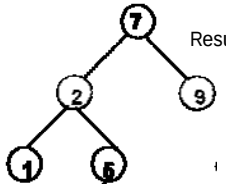


Mediana en ABB

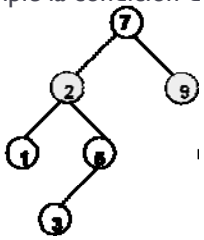
- Se pide diseñar en la clase ABB un método esMediana tal que compruebe si un cierto x dado es el elemento mediana de un ABB:
  - Si el número de elementos mayores que x en el ABB es igual al de menores que x.
  - Asúmase en el diseño que x puede o no ser un elemento del ABB y que no hay elementos duplicados.
- Se quiere obtener un método **iterativo** tal que al aplicarse sobre un ABB Equilibrado tenga un coste asintótico máximo, en el peor de los casos, del orden del logaritmo del tamaño del ABB.

Altura de Equilibrado

- Suponiendo que no se dispone del método altura(), se pide diseñar en la clase ABB un método alturaDeEquilibrado tal que:
  - Si el ABB sobre el que se aplica es Equilibrado devuelve su altura y sino lanza la Excepción ABBNoEquilibrado tan pronto se encuentra un Nodo del ABB que incumple la condición de Equilibrio.
- Ejemplos:



Resultado: 2



Resultado: Excepción

- **Nota:** se dice que un ABB es Equilibrado si la diferencia de alturas entre el Hijo Izquierdo y Derecho de cualquiera de sus Nodos es como máximo 1

toString en Rango (I)

Sea el siguiente método recursivo:

```
/** devuelve un String con los datos del nodo actual de un ABB en el intervalo
    no vacío [inf … sup] **/
protected String toStringEnRango(NodoABB<E> actual, E inf, E sup) {
   String res = "";
   if ( actual != null ) {
      int rInf = actual.dato.compareTo(inf);
      int rSup = actual.dato.compareTo(sup);
      res += toStringEnRango(actual.izq, inf, sup);
      if ( rInf >= 0 && rSup <=0 ) res += actual.dato.toString()+"\n";
      res += toStringEnRango(actual.der, inf, sup);
   }
   return res; }
```


toString en Rango (II)

1. Considerando que la talla del problema que resuelve toStringEnRango es x=tamanyo(actual) y que no presenta instancias significativas, **se pide** :
   a) Escribir las Relaciones de Recurrencia que expresan la complejidad del método si actual es un Nodo de un ABB Equilibrado.
   b) Escribir las Relaciones de Recurrencia que expresan la complejidad del método si actual es un Nodo de un ABB Completamente Degenerado.
   c) Escribir las cotas de complejidad temporal asintótica
2. Usando la propiedad de ordenación de un ABB, diseña una solución más eficiente que la dada.