

Tema 4 – S1

Map Ordenado y Árbol Binario de Búsqueda (ABB)

Contenidos

1. El modelo *Map* Ordenado: definición, coste estimado y ejemplos de uso
2. Árbol Binario de Búsqueda (Equilibrado)
 - Estudio del Árbol Binario y las condiciones de Búsqueda eficiente sobre él
 - Estudio del Árbol Binario **de Búsqueda** y la relación entre su eficiencia y grado de equilibrio

1. Map Ordenado

Modelo

Un Map Ordenado es un conjunto dinámico de **x** Entradas que soportan **eficientemente** no solo las operaciones de un Map sino también las **típicas de una Lista Ordenada** (siguiente en el orden o sucesor, anterior en el orden o predecesor, máximo, mínimo...). Precisamente porque maximiza las ventajas de Map y Lista Ordenada y, a la vez, minimiza sus inconvenientes, el coste máximo estimado de una de sus operaciones básicas es $\log x$, el de la Búsqueda Binaria en un conjunto estático y ordenado

En el estándar Java este modelo es SortedMap, de la Jerarquía Collection

```
package librerias.estructurasDeDatos.modelos;

public interface MapOrdenado<C extends Comparable<C>, V> extends Map<C, V> {
    EntradaMap<C, V> recuperarEntradaMin(); // Entrada de clave mínima
    C recuperarMin(); // clave máxima
    EntradaMap<C, V> recuperarEntradaMax(); // Entrada de clave mínima
    C recuperarMax(); // clave máxima
    EntradaMap<C, V> sucesorEntrada(C c); // siguiente Entrada a c en orden
    C sucesor(C c); // siguiente clave a c en el orden
    EntradaMap<C, V> predecesorEntrada(C c); // Entrada anterior a c en orden
    C predecesor(C c); // clave anterior a c en el orden
    ...
}
```

INDISPENSABLE garantizar la consistencia de los métodos equals y compareTo para cualquier par de claves c1 y c2:

$c1.compareTo(c2) == 0$ **SII** $c1.equals(c2)$

1. Map Ordenado

Ejemplos de uso



Problema 1: Listar las Entradas de un Map en orden Asc. (Desc.)

Diseña un método estático, genérico e iterativo `entradas` que devuelva una `ListaConPI` con las Entradas de un Map `m` ordenadas Asc. (Desc.)

Problema 2: Ordenación

Diseña un método estático, genérico e iterativo `mapSort` que, con la ayuda de un `MapOrdenado`, ordene los elementos (`Comparable`) de un array `v`

Problema 3: 2-SUM (simplificación del problema de la suma de subconjuntos)

Diseña un método estático, genérico e iterativo `hayDosQueSuman` que, dados un array `v` de enteros y un entero `k`, determine si existen en `v` dos números cuya suma sea `k`; usa un `Map Ordenado` como EDA auxiliar

2. Árbol Binario de Búsqueda (Equilibrado)

¿Qué Representación usar para Map Ordenado? ¿Soporta el coste estimado ($\log x$ en promedio) de sus operaciones básicas?

| Representación \ Coste Promedio | recuperar | | insertar eliminar | | sucesor predecesor | | eliminarMin eliminarMax | | recuperarMin recuperarMax | |
|---------------------------------|------------------|--|----------------------------|--|-----------------------|--|----------------------------|--|------------------------------|--|
| | | | | | | | | | | |
| Lineal | $\Theta(x)$ | | $\Theta(1)$ $\Theta(x)$ | | $\Theta(x)$ | | $\Theta(x)$ | | $\Theta(x)$ | |
| Lineal Contigua Ordenada | $\Theta(\log x)$ | | $\Theta(x)$ | | $\Theta(\log x)$ | | $\Theta(1)$ | | $\Theta(1)$ | |
| Tabla Hash | $\Theta(1)$ | | $\Theta(1)$ | | $\Theta(x)$ | | $\Theta(x)$ | | $\Theta(x)$ | |

¿Representación Jerárquica o en Árbol?

¿Qué tipo de Árbol?

¿Condiciones para la Búsqueda eficiente en ellos?

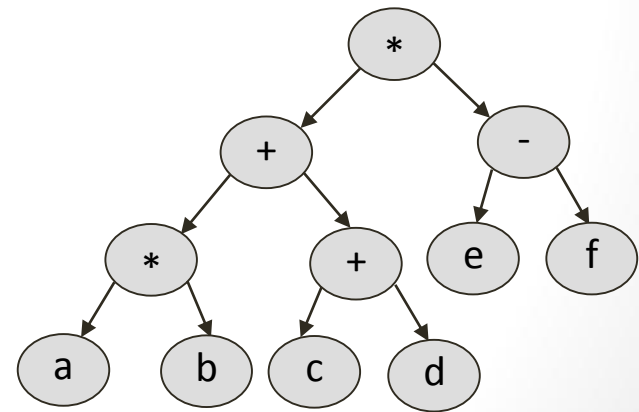
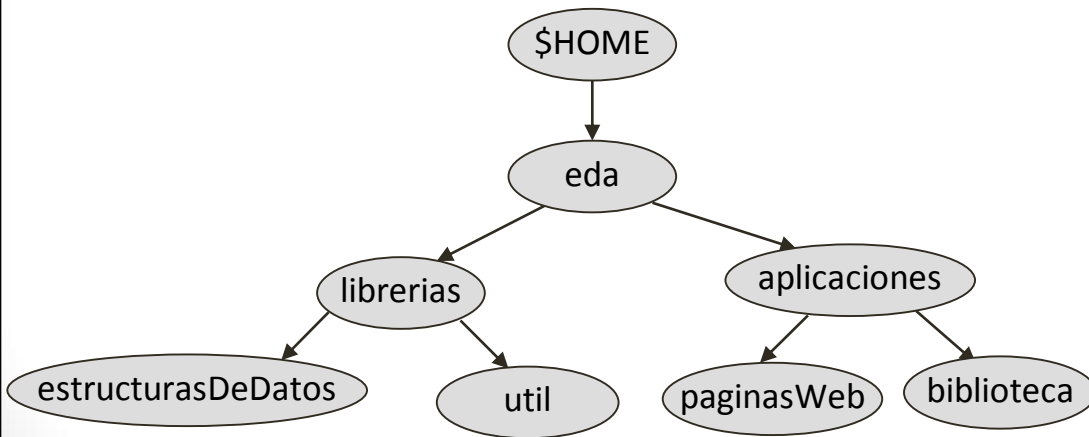
2. ABB (Equilibrado)

Modelos Lineales VS Modelos Jerárquicos

- Con una Representación Lineal de los datos (solo) podemos representar relaciones muy simples entre ellos: **elSiguiente** o **elAnterior**
- Con una Representación Jerárquica, o en Árbol, de los datos podemos representar relaciones más complejas, jerárquicas, entre ellos: **padreDe**, **hijoDe**, **ascendienteDe**, **descendienteDe**, etc.

Estructura de directorios de las prácticas de EDA

Expresión aritmética $((a*b)+(c+d))*(e-f)$



2. ABB (Equilibrado)

Estudio del Árbol Binario

Para que te familiarices con los conceptos básicos sobre Árbol y Árbol Binario (AB) (definiciones, nomenclatura, propiedades, operaciones y sus costes) hemos preparado un cuestionario en PoliformaT

Ejercicio 2: Examen poli [formaT]

“Tema 4 - S1: Cuestiones sobre Árboles Binarios”

Para resolverlo, puedes consultar el documento...

“Apuntes –Introducción a la Representación Jerárquica de una EDA: Árbol Binario”

también disponible en PoliformaT (consulta la guía didáctica para más información)

2. ABB (Equilibrado)

Condiciones para la Búsqueda eficiente en un AB

Dado que...

- a) En promedio y en el peor caso, el coste Temporal Asintótico de la Búsqueda en un AB de talla N es $O(N)$ (asumiendo que el tiempo que se tarda en visitar un Nodo es constante)

Independientemente de su grado de equilibrio, se recorren todos los caminos del AB desde la Raíz a las Hojas y, por tanto, se visitan TODOS sus Nodos

- b) En un AB Equilibrado de tamaño N el coste Temporal Asintótico del Recorrido de su Camino Más Largo es, como máximo, $\log N$, su altura máxima

... Para que la Búsqueda en un AB tenga un coste subLineal se requiere:

1. PROPIEDAD ESTRUCTURAL: AB Equilibrado
2. PROPIEDAD DE ORDENACIÓN: datos en cierto Orden para poder localizarlos

Comparable

- Bien explorando UN ÚNICO CAMINO del AB, en vez de todos
- Bien accediendo directamente a UN SOLO NODO del AB, en vez de a todos

2. ABB (Equilibrado)

¿Qué tipo de Árboles Binarios usar para Representar un Map Ordenado?

| Coste Promedio Representación | recuperar | insertar eliminar | sucesor predecesor | eliminarMin eliminarMax | recuperarMin recuperarMax |
|--|------------------|----------------------------|-----------------------|----------------------------|------------------------------|
| Lineal | $\Theta(x)$ | $\Theta(1)$ $\Theta(x)$ | $\Theta(x)$ | $\Theta(x)$ | $\Theta(x)$ |
| Lineal Contigua Ordenada | $\Theta(\log x)$ | $\Theta(x)$ | $\Theta(\log x)$ | $\Theta(1)$ | $\Theta(1)$ |
| Tabla Hash | $\Theta(1)$ | $\Theta(1)$ | $\Theta(x)$ | $\Theta(x)$ | $\Theta(x)$ |
| Árbol Binario de Búsqueda Equilibrado ? | $\Theta(\log x)$ | $\Theta(\log x)$ | $\Theta(\log x)$ | $\Theta(\log x)$ | $\Theta(\log x)$ |

AB Equilibrado & Parcialmente Ordenado

2. ABB (Equilibrado)

Definición

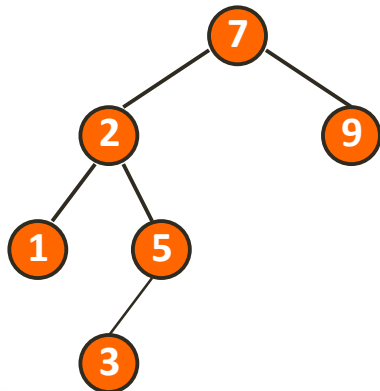
Un Árbol Binario de Búsqueda (ABB) Equilibrado es un Árbol Binario (AB) tal que...

1. PROPIEDAD ESTRUCTURAL: AB Equilibrado

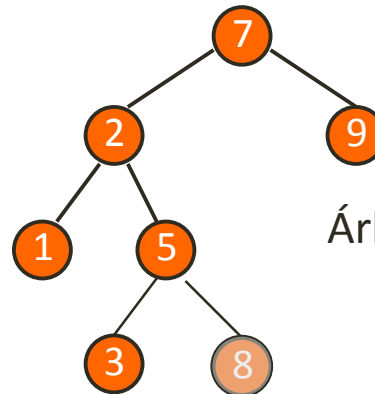
2. PROPIEDAD DE ORDENACIÓN: Propiedad de Búsqueda Ordenada?

Un AB es **de Búsqueda** o cumple la **propiedad de Búsqueda Ordenada** si:

- Todos los datos de su subÁrbol Izquierdo son menores -o iguales- que el que ocupa su Nodo Raíz
- Todos los Datos de su subÁrbol Derecho son mayores que el que ocupa su Nodo Raíz
- Sus subÁrboles Izquierdo y Derecho también son ABB



Árbol Binario *de Búsqueda*



Árbol Binario, *¡a secas!*

2. ABB (Equilibrado)

Para que te familiarices con la definición de un ABB, sus operaciones y propiedades, hemos preparado un cuestionario en PoliformaT

Ejercicio 3: Examen poli **[formaT]**

“Tema 4 - S1: Cuestiones sobre ABBs y sus propiedades”

Para resolver este examen, puede serte de utilidad el video [El Árbol Binario de Búsqueda \(G. Moltó 2010\)](#)

También hemos preparado este ejercicio CAP:

Ejercicio 4: ABB de Aristas



(clave: CCDIH00Z)

La clase *Arista* representa una arista de un grafo como un triplete (origen, destino, peso). Complétala para que el conjunto de aristas de un grafo se pueda representar mediante un ABB ordenado por pesos

2. ABB (Equilibrado)

Propiedades y operaciones

- **Localización de un Dato d en $O(H)$, i.e. $O(\log N)$ en un ABB Equilibrado**

pues sólo se debe explorar en Profundidad **uno** de sus Caminos desde la Raíz

→ Operaciones en las que es posible: `recuperar(e)`, `insertar(e)`, `eliminar(e)`, `recuperarMin()`, `eliminarMin()`, `recuperarMax()`, `eliminarMax()`, `sucesor(e)`, `predecesor(e)`, etc.

- **Ordenación Ascendente de sus datos en $\Theta(N)$** , pues basta recorrerlo In-Orden

- **Cálculo del tamaño de un ABB y el de cada uno de sus Nodos en $\Theta(1)$** , pues al existir un único Camino de inserción y borrado se puede calcular el tamaño de sus Nodos conforme se va construyendo el ABB

- al insertar el Elemento e se crea el Nodo Hoja que lo contiene con tamaño 1
- al insertar/eliminar un Descendiente propio de un Nodo dado de un ABB su tamaño se incrementa/decrementa en 1

→ **Selección del k -ésimo menor elemento de una Colección en $O(H)$**

→ **Operaciones de rango en $O(H)$ i.e. $O(\log N)$ en un ABB Equilibrado**

2. ABB (Equilibrado)

Eficiencia y Grado de Equilibrio: ¿es Equilibrado de per se un ABB?

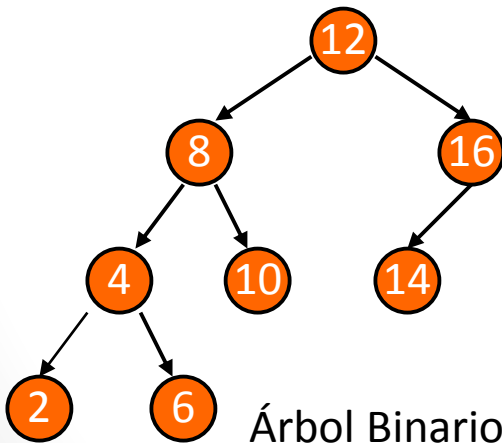
Un Árbol Binario de Búsqueda (ABB) Equilibrado es un Árbol Binario (AB) tal que...

1. PROPIEDAD ESTRUCTURAL?

AB Equilibrado

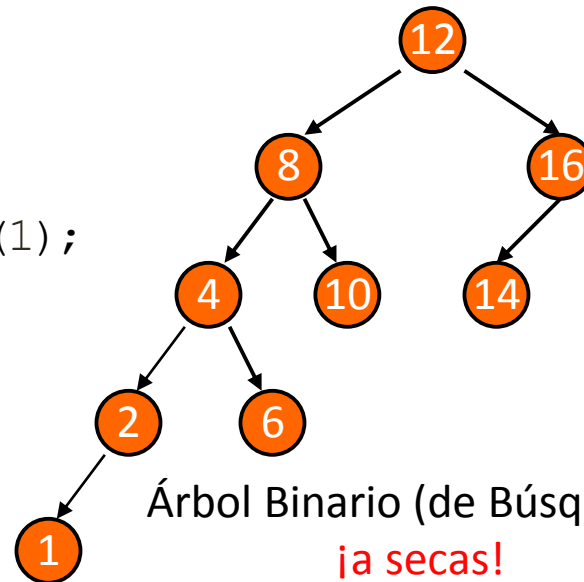
2. PROPIEDAD DE ORDENACIÓN:

Propiedad de **Búsqueda Ordenada**, lo que permite encontrar **CUALQUIER** dato del AB explorando tan solo **UNO** de sus caminos



Árbol Binario (de Búsqueda)
Equilibrado

`insertar(1);`



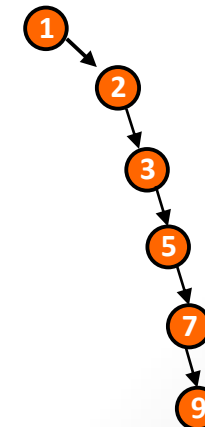
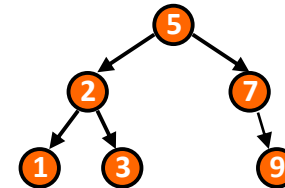
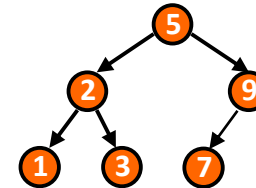
Árbol Binario (de Búsqueda)
¡a secas!

2. ABB (Equilibrado)

Eficiencia y Grado de Equilibrio: ¿de qué dependen?

El **grado de equilibrio de un ABB**, o lo uniforme que resulte su distribución Por Niveles, **depende fuertemente del orden de inserción de sus datos** (como en *Quick Sort*). Por ejemplo, si los datos que contiene un ABB son 1, 2, 3, 5, 7 y 9

- Si su secuencia de inserción es 5, 2, 1, 3, 9 y 7 se obtiene un ABB... ¡Completo!, luego Equilibrado
- Si su secuencia de inserción es 5, 2, 7, 1, 3 y 9 se obtiene un ABB... ¡Equilibrado!, pero no Completo
- Si su secuencia de inserción es 1, 2, 3, 5, 7, 9 se obtiene un ABB... ¡Completamente Degenerado!



2. ABB (Equilibrado)

Eficiencia y Grado de Equilibrio: tipos de ABB

Equilibrado de per se

Para imponer la condición de Equilibrio...

- (1) **ABB Bien Equilibrado**, o ABB al que se impone una Condición de Equilibrio tal que $H \approx \lfloor \log N \rfloor$, como por ejemplo los **Árboles “AVL”** (Adelson, Velskii y Landis) o los **“Rojinegros”**

→ Ello obliga a sofisticar la Representación del ABB y la implementación de sus operaciones de inserción y eliminación

- (2) **Construir un ABB Equilibrado a partir de un array v que contenga los datos a insertar ordenados Ascendentemente**, como sigue:

2.1. El dato en $v[\text{mitad}]$, la mediana de los datos de v , se sitúa como Raíz del ABB en construcción

2.2.- Se construye el Hijo Izquierdo del ABB aplicando recursivamente el punto **2.1** a los datos del (sub)array $v[0, \text{mitad} - 1]$ hasta vaciarlo

2.3.- Se construye el Hijo Derecho del ABB aplicando recursivamente el punto **2.1** a los Datos del (sub)array $v[\text{mitad} + 1, v.\text{length} - 1]$ hasta vaciarlo

Tema 4 – S2

Map Ordenado y Árbol Binario de Búsqueda

Contenidos

3. Implementación de un ABB: las clases ABB y NodoABB

- Representación en memoria: atributos de las clases
- Implementación de sus operaciones: diseño y análisis del coste de los métodos de la clase ABB
 - recuperar, insertar, eliminar, recuperarMin y eliminarMin

Ejercicios

3. Implementación de un ABB

Clases NodoABB y ABB: estructura y métodos básicos

```
package librerias.estructurasDeDatos.jerarquicos;

public class ABB<E extends Comparable<E>> {
    protected NodoABB<E> raiz;

    public ABB() { this.raiz = null; }
    public boolean esVacio() { return this.raiz == null; }
    public talla() { return talla(this.raiz); }
    protected talla(NodoABB<E> actual) {
        if (actual == null) { return 0; }
        else { return actual.talla; }
    }
    ...
}

class NodoABB<E> {
    protected E dato;
    protected NodoABB<E> izq, der;
    int talla;

    NodoABB(E e) {
        this.dato = e;
        this.izq = null; this.der = null;
        talla = 1;
    }
}
```


3. Implementación de un ABB

Métodos de la clase ABB: recuperar de un nodo !ok

Cuestión: el siguiente método recursivo permite **recuperar** de forma **INEFICIENTE** un dato e del Nodo actual de un ABB, pues su diseño ignora su propiedad de Búsqueda Ordenada

```
protected NodoABB<E> recuperar(E e, NodoABB<E> actual) {  
    NodoABB<E> res = actual; // Caso base: equivalente a res = null;  
    if (actual != null) {  
        if (!actual.dato.equals(e)) {  
            res = recuperar(e, actual.izq);  
            if (res == null) { res = recuperar(e, actual.der); }  
        }  
        // else NO hacer nada porque res se ha inicializado a actual  
    }  
    return res;  
}
```

¿Qué cambios se deben realizar en su código para, aprovechando esta propiedad, mejorar la eficiencia del método?

3. Implementación de un ABB

*Métodos de la clase ABB: recuperar de un Nodo **ok!***

Cuestión: el siguiente método recursivo permite **recuperar...**

¿Qué cambios se deben realizar en su código para, aprovechando esta propiedad, mejorar la eficiencia del método?

```
protected NodoABB<E> recuperar(E e, NodoABB<E> actual) {  
    NodoABB<E> res = actual; // Caso base: equivalente a res = null;  
    if (actual != null) {  
  
        int resC = actual.dato.compareTo(e);  
        if (resC > 0) { res = recuperar(e, actual.izq); }  
        else if (resC < 0) { res = recuperar(e, actual.der); }  
        // else NO hacer nada porque res se ha inicializado a actual  
    }  
    return res;  
}
```

Solución: **comparar** cada dato de cada Nodo del ABB con e, de forma que...

- Si el dato del Nodo es mayor que e SOLO se puede seguir buscando en el Hijo Izquierdo del Nodo, pues seguro que en el Derecho no está
- Sino, si el dato del Nodo es menor que e SOLO hay que buscar en el Hijo Derecho del Nodo, pues seguro que en el Izquierdo no está
- Sino, si el dato del Nodo es igual que e, se ha encontrado el dato a recuperar

**Reducción
Logarítmic
a**

3. Implementación de un ABB

Métodos de la clase ABB: recuperar de un ABB

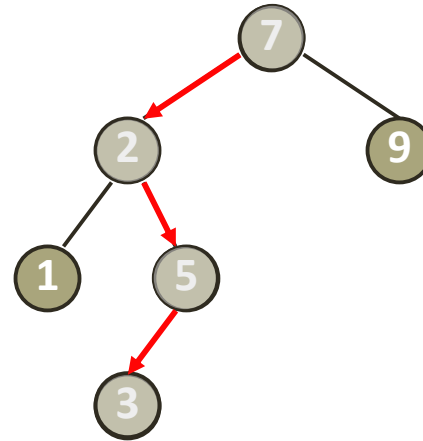
```
protected NodoABB<E> recuperar(E e, NodoABB<E> actual) {  
    NodoABB<E> res = actual;  
    if (actual != null) {  
        int resC = actual.dato.compareTo(e);  
        if (resC > 0) { res = recuperar(e, actual.izq); }  
        else if (resC < 0) { res = recuperar(e, actual.der); }  
        // else NO hacer nada porque res se ha inicializado a actual  
    }  
    return res;  
}
```

```
public E recuperar(E e) {  
    NodoABB<E> res = recuperar(e, this.raiz);  
    if (res == null) { return null; }  
    else { return res.dato; }  
}
```

3. Implementación de un ABB

Métodos de la clase ABB: coste de recuperar

recuperar(e), con $e = 3$



Búsqueda en Pre-Orden de e en un único Camino del ABB (tamaño N y altura H)

- Si e está en el Nodo Raíz del ABB $\rightarrow T_{\text{recuperar}}(x=N) \in \Omega(1)$
- Si e no está en el ABB y se busca en su Camino Más Largo...

(a) ABB CD: $T_{\text{recuperar}}(N) \in O(N = H)$

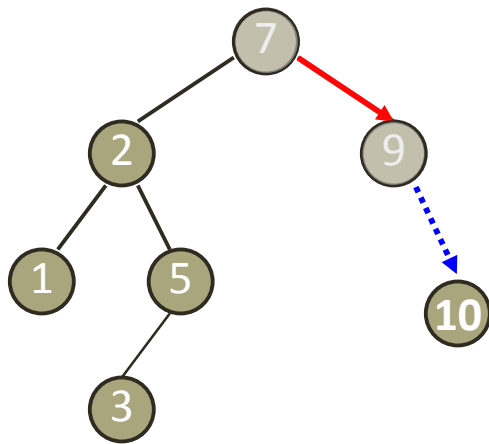
(b) ABB Equilibrado: $T_{\text{recuperar}}(N) \in O(\log N = H)$

$$T_{\text{recuperar}}^{\mu}(N) \in O(\log N = H)$$

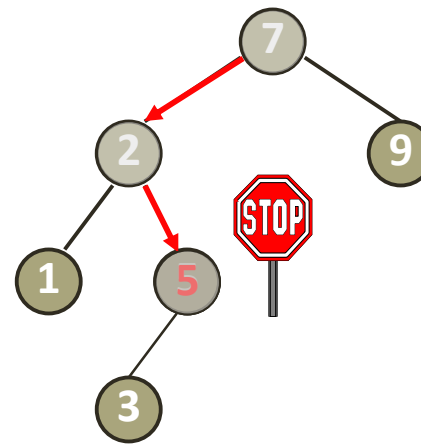
3. Implementación de un ABB

Métodos de la clase ABB: insertar (sin duplicados) en un ABB y su coste

insertar(10), que NO está



insertar(5), que SÍ está



Operación en **2** pasos:

1. Búsqueda del *lugar de inserción de e* en un único Camino del ABB
2. Resolución de la Búsqueda: si *e* está **actualizar** y si no está **insertar** una nueva Hoja que lo contenga

$$T^{\mu}_{\text{insertar}}(N) \in O(\log N = H)$$

3. Implementación de un ABB

Métodos de la clase ABB: diferencias (en verde) entre insertar y recuperar de un Nodo

```
protected NodoABB<E> recuperar(E e, NodoABB<E> actual) {  
    NodoABB<E> res = actual;  
    if (actual != null) {  
        int resC = actual.dato.compareTo(e);  
        if (resC > 0) { res = recuperar(e, actual.izq); }  
        else if (resC < 0) { res = recuperar(e, actual.der); }  
        // else NO hacer nada porque res se ha inicializado a actual  
    }  
    return res;  
}
```

```
protected NodoABB<E> insertar(E e, NodoABB<E> actual) {  
    NodoABB<E> res = actual;  
    if (actual != null) {  
        int resC = actual.dato.compareTo(e);  
        if (resC > 0) { res.izq = insertar(e, actual.izq); }  
        else if (resC < 0) { res.der = insertar(e, actual.der); }  
        else { res.dato = e; }  
        res.talla = 1 + talla(res.izq) + talla(res.der);  
    }  
    Se actualiza la talla ¿Por qué al resolver la llamada?  
    else { res = new NodoABB<E>(e); }  
    Se inicializa la talla a 1  
    return res;  
}
```

3. Implementación de un ABB

*Métodos de la clase ABB: diferencias (**en verde**) entre insertar y eliminar de un ABB*

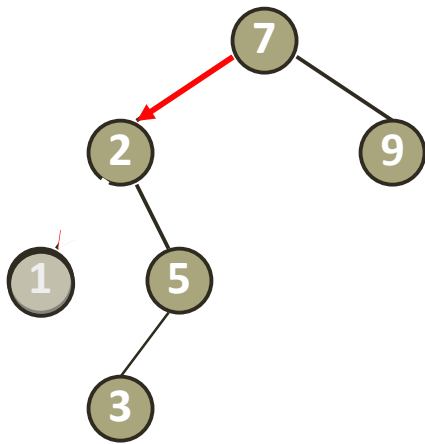
```
public void insertar(E e) {  
    // devuelve el Nodo Raíz que se le pasa como parámetro,  
    // con un Nodo más si e NO estaba en él  
    this.raiz = insertar(e, this.raiz)  
}  
  
public void eliminar(E e) {  
    // devuelve el Nodo Raíz que se le pasa como  
    // parámetro, con un Nodo menos si e SÍ estaba en él  
    this.raiz = eliminar(e, this.raiz)  
}
```

¿De qué sitio del ABB y cómo eliminar un dato?

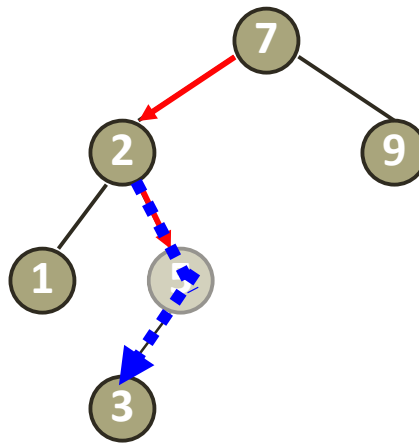
3. Implementación de un ABB

Métodos de la clase ABB: diferencias (en verde) entre insertar y eliminar de un ABB. Coste de eliminar

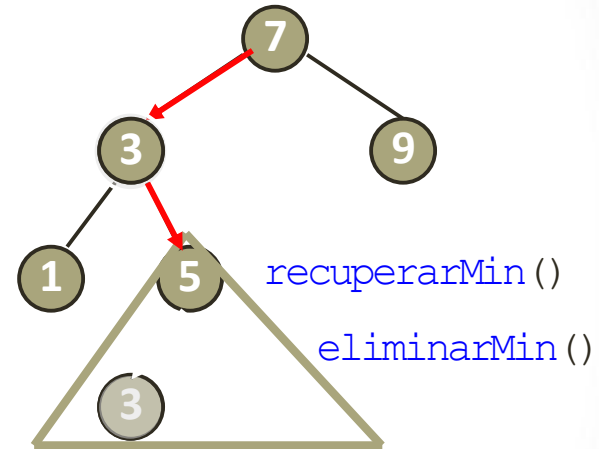
eliminar(1),
en una **Hoja**



eliminar(5),
en un **Nodo con 1 Hijo**



eliminar(2),
en un **Nodo con 2 Hijos**



Operación en **2** pasos:

1. **Búsqueda** *de e* en un único Camino del ABB
2. **Resolución de la Búsqueda**: si *e* está, **borrar** el Nodo que lo contiene y si no está **NO hacer nada**


$$T_{\text{eliminar}}^{\mu}(N) \in O(\log N)$$

3. Implementación de un ABB

Métodos de la clase ABB: diferencias (en verde) entre insertar y eliminar de un Nodo

```
protected NodoABB<E> insertar(E e, NodoABB<E> actual) {
    NodoABB<E> res = actual;
    if (actual != null) {
        int resC = actual.dato.compareTo(e);
        if (resC > 0) { res.izq = insertar(e, actual.izq); }
        else if (resC < 0) { res.der = insertar(e, actual.der); }
        else { res.dato = e; }
        res.talla = 1 + talla(res.izq) + talla(res.der);
    }
    else { res = new NodoABB<E>(e); }
    return res;
}

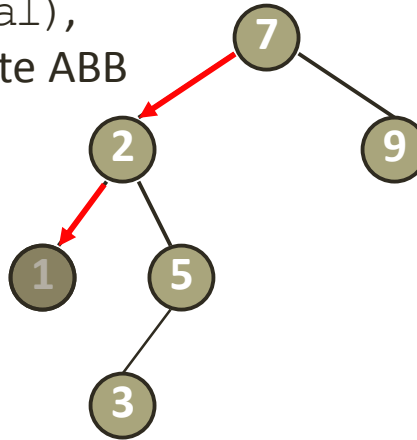
protected NodoABB<E> eliminar(E e, NodoABB<E> actual) {
    NodoABB<E> res = actual;
    if (actual != null) {
        int resC = actual.dato.compareTo(e);
        if (resC > 0) { res.izq = eliminar(e, actual.izq); }
        else if (resC < 0) { res.der = eliminar(e, actual.der); }
        else { // eliminar actual
            if (actual.izq == null) { return actual.der; }
            else if (actual.der == null) { return actual.izq; }
            else {
                res.dato = recuperarMin(actual.der).dato;
                res.der = eliminarMin(actual.der);
            }
            res.talla = 1 + talla(res.izq) + talla(res.der);
        }
    }
    return res;
}
```

 **Se actualiza la talla** ¿También si tiene 1 o 0 Hijos?

3. Implementación de un ABB

Métodos de la clase ABB: recuperarMin de un Nodo y de un ABB. Coste de recuperarMin

Ejemplo: `recuperarMin(actual)`,
siendo actual el Nodo Raíz de este ABB



Recorrido en Pre-Orden del Camino más a la izquierda del ABB hasta alcanzar su último Nodo, que puede no ser una Hoja, y devolverlo: $T^{\mu}_{\text{recuperarMin}}(N) \in O(\log N)$

//SII actual != null: devuelve el Nodo de actual que contiene a su mínimo

```
protected NodoABB<E> recuperarMin(NodoABB<E> actual) {  
    NodoABB<E> res = actual;  
    if (actual.izq != null) { res = recuperarMin(actual.izq); }  
    return res;  
}
```

```
// SII !esVacio()
```

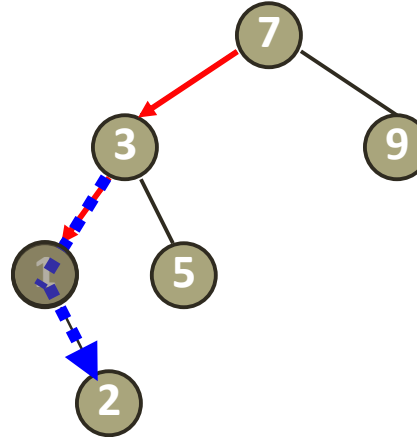
```
public E recuperarMin() { return recuperarMin(raiz).dato; }
```

¡SIN usar compareTo!

3. Implementación de un ABB

Métodos de la clase ABB: diferencias (en verde) entre recuperarMin y eliminarMin de un Nodo/ABB. Coste de eliminarMin

Ejemplo: `eliminarMin(actual)`,
siendo actual el Nodo Raíz de este ABB



Recorrido en Pre-Orden del Camino más a la izquierda del ABB hasta alcanzar su último Nodo, que puede no ser una Hoja, y **borrarlo**: $T^{\mu}_{\text{eliminarMin}}(N) \in O(\log N)$

```
//SII actual != null: devuelve el Nodo actual tras eliminar su mínimo
protected NodoABB<E> eliminarMin(NodoABB<E> actual) {
    NodoABB<E> res = actual;
    if (actual.izq != null) {
        res.izq = eliminarMin(actual.izq);
        res.talla--; // o res.talla = 1 + talla(res.izq) + talla(res.der);
    }
    else { res = actual.der; } // SII !esVacio()
    return res;
}
```

¡SIN usar compareTo!

```
public E eliminarMin() {
    E res = recuperarMin();
    this.raiz = eliminarMin(this.raiz);
    return res;
}
```

¿Dos veces?

3. Implementación de un ABB

Para que te familiarices con la definición de los métodos de ABB hemos preparado los siguientes cuestionarios en PoliformaT

Ejercicio 5: Examen poli **[formaT]**

“Tema 4 – S2 - Parte 1: Cuestiones sobre los métodos de un ABB y sus costes”

Ejercicio 6: Examen poli **[formaT]**

“Tema 4 – S2 - Parte 2: Cuestiones sobre las trazas de algunos métodos de ABB”

También hemos preparado este ejercicio CAP:

Ejercicio 7: Comprobar si e es una Hoja de un ABB y su coste



(clave: CCDIG00Z)

Diseña en la clase ABB el método `esHoja`, que comprueba si e está en una Hoja de un ABB (sin repetidos) tras efectuar el mínimo número de `compareTo` posibles

Tema 4 – S3

Map Ordenado y Árbol Binario de Búsqueda

Contenidos

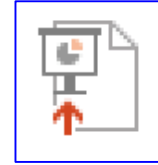
3. Implementación de un ABB: las clases `ABB` y `NodoABB`

- Representación en memoria: atributos de las clases
- Implementación de sus operaciones: diseño y análisis del coste de los métodos de la clase `ABB`
 - sucesor

Ejercicios

3. Implementación de un ABB

Métodos de la clase ABB: sucesor
(diferencias con recuperar *en verde*)



```
public E sucesor(E e) {
    NodoABB<E> res = sucesor(e, this.raiz);
    if (res == null) { return null; }
    else { return res.dato; }
}

protected NodoABB<E> sucesor(E e, NodoABB<E> actual) {
    NodoABB<E> res = null;
    if (actual != null) {
        int resC = actual.dato.compareTo(e);
        if (resC > 0) {
            res = sucesor(e, actual.izq); //va a la izq
            //vuelve de la izq, donde siempre está el sucesor
            if (res == null) { res = actual; } //actualiza sucesor
        }
        else { res = sucesor(e, actual.der); } //va a la dra
            //vuelve de la dra, luego el sucesor no varía
    }
    return res;
}
```

3. Implementación de un ABB

Ejercicios



Ejercicio 8: el siguiente método devuelve el nivel en el que se encuentra la primera aparición en Pre-Orden de *e* en un ABB Equilibrado, o -1 si no está

```
public int enQueNivel(E e) {  
    return enQueNivel(e, this.raiz, 0);  
}  
  
protected int enQueNivel(E e, NodoABB<E> actual, int nivelActual) {  
    int res = -1;  
    if (actual != null) {  
        if (actual.dato.equals(e)) { res = nivelActual; }  
        else {  
            res = enQueNivel(e, actual.izq, nivelActual + 1);  
        }  
        if (res == -1) {  
            res = enQueNivel(e, actual.der, nivelActual + 1);  
        }  
    }  
    return res;  
}
```

(a) Analiza su coste; supón que *N*, el tamaño del ABB, es la talla del problema

(b) Usando la propiedad de Ordenación de un ABB, modifica el código del método recursivo para mejorar su coste. Además, analiza el coste del método resultante para comprobar que, en efecto, es el mínimo posible

3. Implementación de un ABB

Ejercicios

Pistas para la resolución del ejercicio 8 (b):

Esqueleto de la versión eficiente del método:

```
protected int enQueNivel(E e, NodoABB<E> actual, int nivelActual) {
    int res = -1;
    if (actual != null) {
        int resC = actual.dato./*COMPLETAR*/;

        if (resC /*COMPLETAR*/) { res = nivelActual; }
        else if (resC /*COMPLETAR*/) {
            res = enQueNivel(e, actual.izq, nivelActual + 1);
        }
        else {
            res = enQueNivel(e, actual.der, nivelActual + 1);
        }
    }
    return res;
}
```


3. Implementación de un ABB

Ejercicios

Ejercicio 9:

(a) Analiza el coste del siguiente método, que devuelve el nº de elementos de un Nodo de un ABB Equilibrado que son mayores que uno dado e

```
protected int contarMayoresQue(E e, NodoABB<E> actual) {  
    int res = 0;  
    if (actual != null) {  
        if (actual.dato.compareTo(e) > 0) { res += 1; }  
        res += contarMayoresQue(e, actual.izq)  
        res += contarMayoresQue(e, actual.der);  
    }  
    return res;  
}
```

(b) Utilizando la propiedad de Ordenación de un ABB, y asumiendo que no existen duplicados, modifica el código presentado para que se ejecute con coste mínimo sobre un objeto de la clase ABB

PISTA: recuerda que usas un ABB con Rango, no un simple ABB

3. Implementación de un ABB

Ejercicios

Pistas para la resolución del ejercicio 9 (b):

Esqueleto de la versión eficiente del método, aplicando tan solo la propiedad de Búsqueda Ordenada:

```
protected int contarMayoresQue(E e, NodoABB<E> actual) {
    int res = 0;
    if (actual != null) {
        int resC = actual.dato.compareTo(e);
        if (resC == 0) {
            res += /*CORREGIR*/contarMayoresQue(e, actual.der);
        }
        else if (resC < 0) {
            res += contarMayoresQue(e, actual.der);
        }
        else {
            res += 1
                + /*CORREGIR*/contarMayoresQue(e, actual.der);
            + contarMayoresQue(e, actual.izq);
        }
    }
    return res;
}
```

$T_{\text{contarMayoresQue}}(x) \in O(x) \dots$ **PERO** $T_{\text{contarMayoresQue}}(x) \in \Omega(\log x)$

3. Implementación de un ABB

Ejercicio propuesto

Ejercicio 10: modificando donde creas necesario el método `enQueNivel` del ejercicio 8, diseña en la clase `ABB` los siguientes métodos con coste logarítmico:

- (a) `padreDe`, que devuelve el dato en el nodo Padre del que contiene `a`, `null` si no está en el ABB o no tiene Padre
- (b) `hermanoDe`, con coste logarítmico, que devuelve el dato en el nodo Hermano del que contiene `a`, `null` si no está en el ABB o no tiene Hermano

Tema 4 – S4

Map Ordenado y Árbol Binario de Búsqueda

Contenidos

4. Ejercicios de diseño de métodos de la clase ABB



Ejercicios

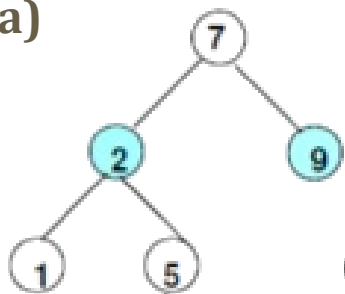
En la clase ABB, diseña un método...

Ejercicio 11: `toListaConPI` que devuelve una Lista Con PI con los datos de un ABB en In-Orden, i.e. ordenados Asc.

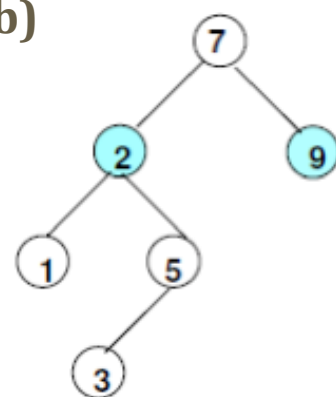
Ejercicio 12: `datosEnNivel` que devuelve un String con los datos del nivel `k` de un ABB; analiza también su coste

Ejercicio 13: `alturaDeEquilibrado`, que devuelve la altura de un ABB si es Equilibrado, o lanza la Excepción `NoSuchElementException` tan pronto encuentra un nodo que incumple la condición de Equilibrio. Por ejemplo, el resultado del método es 2 si se aplica al ABB de la Figura (a); sin embargo, lanza `NoSuchElementException` si se aplica al ABB de la Figura (b)

Figura (a)



Figura(b)



NOTA: comenta el método `altura` que hay en la clase ANTES de implementar este método.

Ejercicios

Ejercicio 14: Dato en nivel



(clave: CCDII00Z)

Diseña un método en la clase `ABB` que compruebe, con el menor número de comparaciones posibles, si un elemento e está en el nivel k de un `ABB`; asume que NO hay elementos repetidos en el `ABB` y que k es un nivel válido