Estructuras de datos y algoritmos

UNIVERSIDAD DE CANTABRIA

1. Introducción
2. Estructuras de datos lineales
3. ***Estructuras de datos jerárquicas***
4. Grafos y caminos
5. Implementación de listas, colas, y pilas
6. Implementación de mapas, árboles, y grafos

UNIVERSIDAD DE CANTABRIA

3. Estructuras de datos jerárquicas

- 3.1 Árboles
- 3.2 Recorrido y ordenación
- 3.3 El ADT árbol
- 3.4 Árboles binarios
- 3.5 Búsqueda

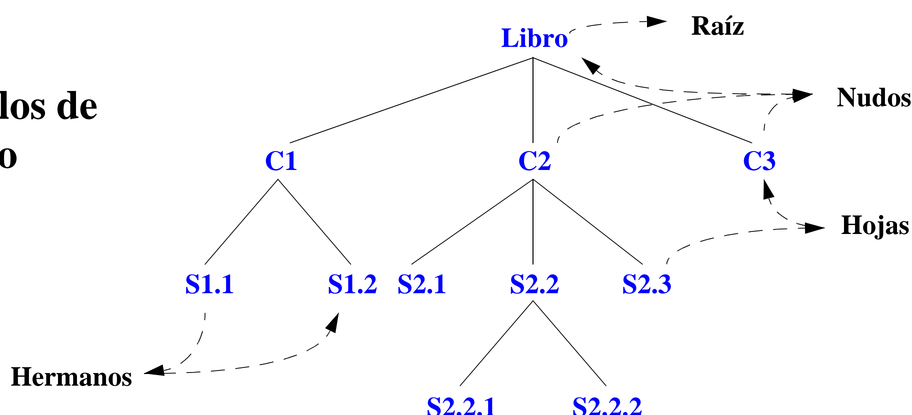
3.1 Árboles

Un árbol es una estructura de datos jerarquizada

Cada dato reside en un nudo, y existen relaciones de parentesco entre nudos:

Ejemplo:

Capítulos de un libro



Notas:

Los **árboles constituyen estructuras de datos jerarquizados**, y tienen multitud de aplicaciones, como por ejemplo:

- Análisis de circuitos, Representación de estructuras de fórmulas matemáticas
- Organización de datos en bases de datos
- Representación de la estructura sintáctica en compiladores.
- En muchas otras áreas de las ciencias del computador.

Un **árbol** está constituido por una colección de elementos denominados **nudos**, uno de los cuales se distingue con el nombre **raíz**, junto con una relación de 'parentesco' que establece una estructura jerárquica sobre los nudos. Cada nudo tiene un padre (excepto el raíz) y puede tener cero o más hijos. Se denomina **hoja** a un nudo sin hijos. Como ejemplo se muestra en la figura superior la tabla de contenidos de un libro

Definición recursiva de los árboles

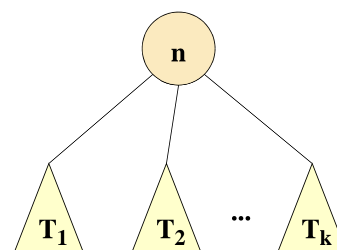
UNIVERSIDAD DE CANTABRIA

Un nudo simple n constituye un árbol

- se denomina la ***raíz*** del árbol

Supongamos que n es un nudo y T_1, T_2, ..., T_k son árboles cuyas raíces son n_1, n_2, ..., n_k, respectivamente.

- Podemos construir un nuevo árbol haciendo que n sea el padre de los nudos n_1, n_2, ..., n_k
- En el nuevo árbol n es la raíz y n_1, n_2, ..., n_k se denominan los hijos de n



Definiciones

UNIVERSIDAD DE CANTABRIA

- ***Camino***: secuencia de nudos tales que cada uno es hijo del anterior
- ***Longitud del camino***: nº de nudos que tiene
- ***Antecesor***: un nudo es antecesor de otro si hay un camino del primero al segundo
- ***Descendiente***: un nudo es descendiente de otro si hay un camino del segundo al primero
- ***Subárbol*** o ***Rama***: Un nudo y todos sus descendientes

3.2 Recorrido y ordenación de los nudos

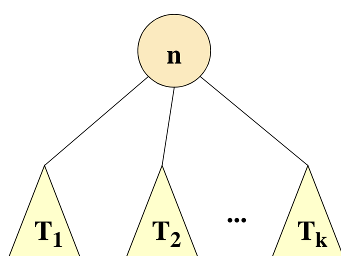
Los hermanos se ordenan generalmente de izquierda a derecha

Dos árboles ordenados, distintos

La ordenación o recorrido de los nudos se suele hacer de 3 modos:

- preorden, postorden, e inorden

Ordenación de los nudos (cont.)



Preorden: $n,T_1,T_2,...,T_k$

Postorden: $T_1,T_2,...,T_k,n$

Inorden: $T_1,n,T_2,...,T_k$

Figura A

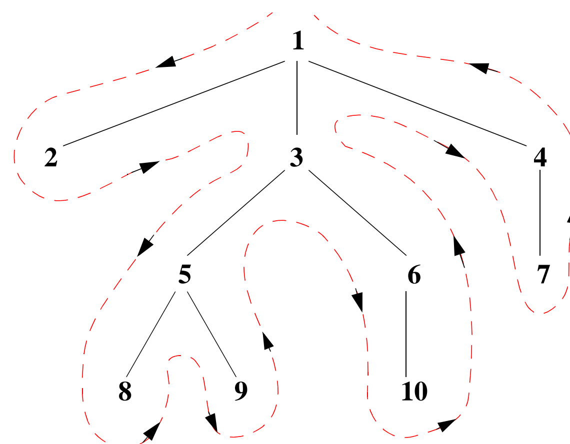


Figura B

Notas:

Estos tipos de ordenación se definen recursivamente de la forma siguiente:

1. Si el árbol T es nulo, entonces la lista vacía es la lista de T en preorden, postorden e inorden.
2. Si T tiene un solo nudo, entonces el nudo es la lista de T en preorden, postorden e inorden.
3. Si T consiste en un árbol con una raíz n y subárboles T_1, T_2, ..., T_k, como en la figura a de arriba:
   - a)La lista de T en preorden es la raíz n seguida de los nudos de T_1 en preorden, luego los nudos de T_2 en preorden, hasta finalizar con la lista de T_k en preorden.
   - b)La lista de T en inorden es la lista de los nudos de T_1 en inorden, seguida de la raíz n, luego los nudos de T_2, ..., T_k con cada grupo de nudos en inorden.
   - c)La lista de T en postorden es la lista de los nudos de T_1 en postorden, luego los nudos de T_2 en postorden, y así hasta la lista de T_k en postorden, finalizando con el nudo raíz n.

Un método para producir estas tres ordenaciones de nudos a mano consiste en recorrer los nudos en la forma que se indica en la figura b de arriba:

- Para ordenar en preorden se lista cada nudo la primera vez que se pasa por él. Para postorden la última vez. Para inorden se listan las hojas la primera vez que se pasa por ellas, pero los nudos interiores la segunda.

Recorrido de árboles

```
método Preorden (N : Nudo; A : Arbol)
  listar N;
  para cada hijo H de N, y empezando por la izquierda
  hacer
    Preorden(H,A);
  fpara;
fmétodo;

método Postorden (N : Nudo; A : Arbol)
  para cada hijo H de N, y empezando por la izquierda
  hacer
    Postorden(H,A);
  fpara;
  listar N;
fmétodo;
```

Recorrido de árboles (cont.)

UNIVERSIDAD DE CANTABRIA

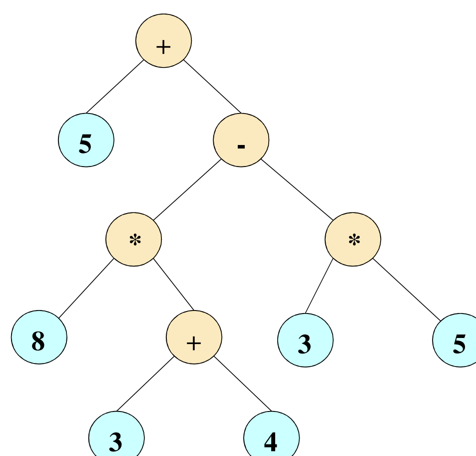
```
método Inorden (N : Nudo; A : Arbol)
  si n es una hoja entonces
    listar n;
  si no
    Inorden(hijo más a la izquierda de n,A);
    listar n;
    para cada hijo h de n, excepto el más a la
        izquierda, y empezando por la izquierda
    hacer
      Inorden(H,A);
    fpara;
  fsi;
fmétodo;
```


Ejemplo de ordenación de expresiones aritméticas

UNIVERSIDAD DE CANTABRIA

Expresión: 5+8*(3+4)-3*5:

- *preorden*: +5-*8+3,4*3,5
- *inorden*: 5+(8*(3+4)-(3*5)) es la expresión en notación matemática normal
- *postorden*: 5,8,3,4+*3,5*-+ es la expresión en Notación Polaca Inversa (RPN)



3.3. El Tipo de datos abstracto árbol

UC UNIVERSIDAD DE CANTABRIA

Operaciones del árbol:

operación	argumentos	retorna	errores
constructor	Elemento	Árbol	
hazNulo			
estaVacio		booleano	
iterador		IteradorDeArbol	

Podemos restringir el árbol a que no esté vacío

- en este caso no lo haremos

© Michael González Harbour

El Iterador de árboles

UC UNIVERSIDAD DE CANTABRIA

La mayoría de las operaciones se encuentran en el iterador de árboles, que

- contiene una referencia a uno de los nudos del árbol
  - inicialmente es la raíz
  - si la referencia es nula, se dice que el iterador *no es válido*
- puede usarse para recorrer y/o modificar el árbol
- si el iterador no es válido, casi todas las operaciones lanzan `NoValido`

© Michael González Harbour

Operaciones del iterador de árboles: operaciones de modificación

UNIVERSIDAD DE CANTABRIA

operación	argumentos	retorna	errores
constructor	elArbol	IteradorDeArbol	
insertaPrimerHijo	Elemento		NoValido
insertaSiguienteHermano	Elemento		EsRaiz, NoValido
eliminaHoja		Elemento	NoEsHoja, NoValido
modificaElemento	Elemento	Elemento viejo	NoValido
cortaRama		Rama cortada	NoValido
reemplazaRama	Nueva Rama	Rama cortada	NoValido
anadeRama	Nueva Rama		NoValido

Notas:

UNIVERSIDAD DE CANTABRIA

- *constructor*: Crea el iterador del árbol, con el nudo actual igual a la raiz, o no válido si el árbol está vacío
- *insertaPrimerHijo*: Añade un hijo al nudo actual, situado más a la izquierda que los actuales, y con el valor indicado
- *insertaSiguienteHermano*: Añade un hijo al padre del nudo actual, situándolo inmediatamente a la derecha del nudo actual. Lanza `EsRaiz` si se intenta añadir un hermano a la raiz
- *eliminaHoja*: Si el nudo actual es una hoja, la elimina del árbol y hace que el nudo actual sea su padre. Si no es una hoja, lanza `NoEsHoja`.
- *modificaElemento*: Modifica el contenido del nudo actual reemplazándolo por el `elementoNuevo`. Retorna el antiguo contenido del nudo actual
- *cortaRama*: Elimina la rama del árbol cuya raíz es en nudo actual, y hace que el nudo actual sea su padre. Retorna la rama cortada como un árbol independiente.
- *reemplazaRama*: reemplaza la rama del árbol cuya raíz es el nudo actual, sustituyéndola por `nuevaRama`; la posición actual no cambia, y será por tanto la raiz de `nuevaRama` en el árbol actual. Retorna la rama que ha sido reemplazada como un árbol independiente.
- *anadeRama*: Añade el árbol indicado por `nuevaRama` haciendo que su raíz sea hija del nudo actual, situándola a la derecha de los hijos actuales, si los hay

Operaciones del iterador de árboles: operaciones de consulta y recorrido

UNIVERSIDAD DE CANTABRIA

operación	argumentos	retorna	errores
irARaiz			
irAPrimerHijo			NoValido
irASiguienteHermano			NoValido
irAPadre			NoValido
contenido		Elemento	NoValido
esHoja		Booleano	NoValido
esRaiz		Booleano	NoValido
esUltimoHijo		Booleano	NoValido
esValido		Booleano	
clonar		IteradorDeArbol	

Notas:

UNIVERSIDAD DE CANTABRIA

- *contenido*: retorna el elemento contenido en el nudo actual
- *iaARaiz*: hace que el nudo actual sea la raíz del árbol; valdrá no válido si el árbol está vacío
- *irAPrimerHijo*: hace que el nudo actual sea el primer hijo del actual; valdrá no válido si el nudo actual no tiene hijos
- *irASiguienteHermano*: hace que el nudo actual sea el siguiente hermano del actual; valdrá no válido si el nudo actual no tiene hermanos derechos
- *irAPadre*: hace que el nudo actual sea el padre del actual; valdrá no válido si el nudo actual era la raiz
- *esHoja*: retorna un booleano que indica si el nudo actual es una hoja o no (es decir si no tiene hijos)
- *esRaiz*: retorna un booleano que indica si el nudo actual es la raíz del árbol
- *esUltimoHijo*: retorna un booleano que indica si el nudo actual es el último hijo de su padre (es decir si no tiene hermanos derechos)
- *esValido*: retorna un booleano que indica si el nudo actual es válido, o no
- *clonar*: retorna un iterador de árbol que es una copia del actual

Interfaz Java para los árboles

```
package adts;


/**
 * Interfaz del ADT árbol
 */

public interface Arbol<E>
{
   IteradorDeArbol<E> iterador();
   void hazNulo();
   boolean estaVacio();
}
```


Interfaz del iterador de árboles

```
package adts;

public interface IteradorDeArbol<E> extends Cloneable
{
  // operaciones de modificación
  void insertaPrimerHijo(E elemento) throws NoValido;
  void insertaSiguienteHermano(E elemento)
        throws EsRaiz, NoValido;
  E eliminaHoja() throws NoEsHoja, NoValido;
  E modificaElemento (E elementoNuevo)
        throws NoValido;
  Arbol<E> cortaRama() throws NoValido;
  Arbol<E> reemplazaRama(Arbol<E> nuevaRama)
        throws NoValido;
  void anadeRama(Arbol<E> nuevaRama) throws NoValido;
```

Interfaz del iterador de árboles (cont.)

UC UNIVERSIDAD DE CANTABRIA

```
    // operaciones de consulta
    E contenido() throws NoValido;
    boolean esHoja() throws NoValido;
    boolean esRaiz() throws NoValido;
    boolean esUltimoHijo() throws NoValido;
    boolean esValido();

    // operaciones de recorrido
    void irARaiz();
    void irAPrimerHijo() throws NoValido;
    void irASiguienteHermano() throws NoValido;
    void irAPadre() throws NoValido;
    //duplicar un iterador
    IteradorDeArbol<E> clone();
}
```


Ejemplos con Árboles

UC UNIVERSIDAD DE CANTABRIA

1. Escribir métodos para recorrer el árbol en preorden e inorden, usando la interfaz Java para el árbol
2. Escribir un programa para crear un árbol que represente una expresión, comenzando por la raíz y descendiendo a las hojas
3. Escribir un método para crear un árbol de expresiones a partir de su descripción en postorden

Ejemplo 1: preorden

```
public static <E> void preorden
     (IteradorDeArbol<E> iterador)
{
   IteradorDeArbol<E> iter= iterador.clone();
   try {
      System.out.print(iter.contenido()+" ");
      iter.irAPrimerHijo();
      while (iter.esValido()) {
         preorden(iter);
         iter.irASiguienteHermano();
      }
   } catch (NoValido e) {
      System.out.println("Error inesperado: "+e);
   }
}
```


Ejemplo 1: postorden

```
public static <E> void postorden
   (IteradorDeArbol<E> iterador)
{
   IteradorDeArbol<E> iter= iterador.clone();
   try {
      E contenidoRaiz=iter.contenido();
      iter.irAPrimerHijo();
      while (iter.esValido()) {
        postorden(iter); iter.irASiguienteHermano();
      }
      System.out.print(contenidoRaiz+" ");
   } catch (NoValido e) {
      System.out.println("Error inesperado: "+e);
   }
}
```

Ejemplo 1: inorden

UNIVERSIDAD DE CANTABRIA

```
public static <E> void inorden
   (IteradorDeArbol<E> iterador)
{
   IteradorDeArbol<E> iter= iterador.clone();
   try {
      E contenidoRaiz=iter.contenido();
      if (iter.esHoja()) {
         System.out.print(contenidoRaiz);
      } else {
         System.out.print("(");
         iter.irAPrimerHijo();
         inorden(iter);
         System.out.print(contenidoRaiz);
         iter.irASiguienteHermano();
```


Ejemplo 1: inorden (cont.)

UNIVERSIDAD DE CANTABRIA

```
         while (iter.esValido()) {
             inorden(iter);
             iter.irASiguienteHermano();
         }
         System.out.print(")");
      }
   } catch (NoValido e) {
      System.out.println("Error inesperado: "+e);
      throw new NullPointerException();
   }
}
```

Ejemplo 2: Creación del árbol

UNIVERSIDAD DE CANTABRIA

Usaremos las clases:

- `Operador`: representa un operador aritmético
- `Variable`: representa un operando variable, con nombre
- `Constante`: representa un operando constante literal, con un valor

La expresión es: 3*x-(base/2)

```
// crear los operadores y operandos
Operador resta=new Operador('-');
Operador mult= new Operador('*');
Operador div=  new Operador('/');
Variable x=new Variable("x",1);
Variable base=new Variable("base",1);
Constante dos=new Constante(2);
Constante tres=new Constante(3);
```


Ejemplo 2: Creación del árbol (cont.)

UNIVERSIDAD DE CANTABRIA

```
Arbol<ElementoDeExpresion> arbol=
      new ArbolCE<ElementoDeExpresion>(resta);
IteradorDeArbol<ElementoDeExpresion> iter=
      arbol.iterador();

iter.insertaPrimerHijo(mult);
iter.irAPrimerHijo();
iter.insertaPrimerHijo(tres);
iter.irAPrimerHijo();
iter.insertaSiguienteHermano(x);
iter.irAPadre(); // mult
```

Ejemplo 2: Creación del árbol (cont.)

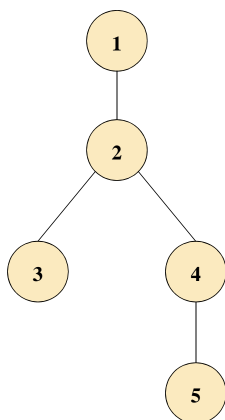

```
iter.insertaSiguienteHermano(div);
iter.irASiguienteHermano();
iter.insertaPrimerHijo(base);
iter.irAPrimerHijo();
iter.insertaSiguienteHermano(dos);
iter.irARaiz();

// mostrar el arbol en preorden
System.out.println("Arbol en preorden:");
OpArboles.preorden(iter);
System.out.println();
```

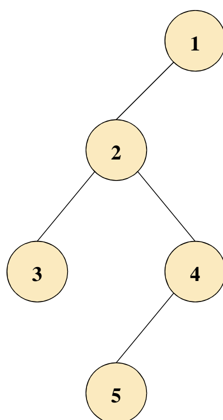

3.4 Árboles binarios

UC UNIVERSIDAD DE CANTABRIA

Un árbol binario es un árbol orientado y ordenado, en el que cada nudo puede tener un hijo izquierdo y un hijo derecho



Un árbol ordinario



Dos árboles binarios

El ADT árbol binario

UNIVERSIDAD DE CANTABRIA

Operaciones del árbol:

operación	argumentos	retorna	errores
constructor		Árbol	
constructor	elemento	Árbol	
constructor	elemento, ramaIzquierda, ramaDerecha	Árbol	
hazNulo			
estaVacio		booleano	
iterador		IterArbolBin	

El iterador de árboles binarios es, conceptualmente idéntico al de los árboles, pero sus operaciones son diferentes

Notas:

UNIVERSIDAD DE CANTABRIA

Las operaciones del ADT arbol son

- *constructor sin parámetros*: Crea un árbol binario vacío
- *constructor con un parámetro*: Crea un árbol binario con un único elemento, que será su raíz
- *constructor con tres parámetros*: Crea un árbol binario cuya raíz es un nudo conteniendo el elemento indicado, y haciendo que su hijo izquierdo sea `ramaIzquierda`, y su hijo derecho sea `ramaDerecha`. Las ramas pueden estar vacías, y en ese caso no se añade hijo izquierdo o derecho, respectivamente

Operaciones del iterador de árboles binarios: operaciones de modificación

UNIVERSIDAD DE CANTABRIA

operación	argumentos	retorna	errores
constructor	elArbolBinario	IterArbolBin	
insertaHijoIzquierdo	Elemento		YaExiste, NoValido
insertaHijoDerecho	Elemento		YaExiste, NoValido
eliminaHoja		Elemento	NoEsHoja, NoValido
modificaElemento	Elemento	Elemento viejo	NoValido
reemplazaRamaIzquierda	Nueva Rama	Rama cortada	NoValido
reemplazaRamaDerecha	Nueva Rama	Rama cortada	NoValido

Notas:

UNIVERSIDAD DE CANTABRIA

- *constructor*: Crea el iterador del árbol, con el nudo actual igual a la raiz, o no válido si el árbol está vacío
- *insertaHijoIzquierdo*: Añade un hijo izquierdo al nudo actual, con el valor indicado. Lanza `YaExiste` si ya existía un hijo izquierdo
- *insertaHijoDerecho*: Añade un hijo derecho al nudo actual, con el valor indicado. Lanza `YaExiste` si ya existía un hijo derecho
- *eliminaHoja*: Si el nudo actual es una hoja, la elimina del árbol y hace que el nudo actual sea su padre. Si no es una hoja, lanza `NoEsHoja`.
- *modificaElemento*: Modifica el contenido del nudo actual reemplazándolo por el `elementoNuevo`. Retorna el antiguo contenido del nudo actual
- *reemplazaRamaIzquierda*: reemplaza la rama del árbol cuya raíz es el hijo izquierdo del nudo actual, sustituyéndola por `nuevaRama` (si es vacía deja el nudo actual sin hijo izquierdo). Retorna la rama que ha sido reemplazada como un árbol independiente (puede ser vacía).
- *reemplazaRamaDerecha*: reemplaza la rama del árbol cuya raíz es el hijo derecho del nudo actual, sustituyéndola por `nuevaRama` (si es vacía deja el nudo actual sin hijo derecho). Retorna la rama que ha sido reemplazada como un árbol independiente (puede ser vacía).

Operaciones del iterador de árboles: operaciones de consulta y recorrido

UNIVERSIDAD DE CANTABRIA

operación	argumentos	retorna	errores
irARaiz			
irAHijoIzquierdo			NoValido
irAHijoDerecho			NoValido
irAPadre			NoValido
contenido		Elemento	NoValido
esHoja		Booleano	NoValido
esRaiz		Booleano	NoValido
tieneHijoIzquierdo		Booleano	NoValido
tieneHijoDerecho		Booleano	NoValido
esValido		Booleano	
clonar		IteradorDeArbol	

Notas:

UNIVERSIDAD DE CANTABRIA

- *contenido*: retorna el elemento contenido en el nudo actual
- *iaARaiz*: hace que el nudo actual sea la raíz del árbol; valdrá no válido si el árbol está vacío
- *irAHijoIzquierdo*: hace que el nudo actual sea el hijo izquierdo del actual; valdrá no válido si el nudo actual no tiene hijo izquierdo
- *irAHijoDerecho*: hace que el nudo actual sea el hijo derecho del actual; valdrá no válido si el nudo actual no tiene hijo derecho
- *irAPadre*: hace que el nudo actual sea el padre del actual; valdrá no válido si el nudo actual era la raiz
- *esHoja*: retorna un booleano que indica si el nudo actual es una hoja o no (es decir si no tiene hijos)
- *esRaiz*: retorna un booleano que indica si el nudo actual es la raíz del árbol
- *tieneHijoIzquierdo*: retorna un booleano que indica si el nudo actual tiene hijo izquierdo o no
- *tieneHijoDerecho*: retorna un booleano que indica si el nudo actual tiene hijo derecho o no
- *esValido*: retorna un booleano que indica si el nudo actual es válido, o no
- *clonar*: retorna un iterador de árbol que es una copia del actual

Interfaz Java para los árboles binarios

```
package adts;

public interface ArbolBinario<E>
{
   IterArbolBin<E> iterador();
   void hazNulo();
   boolean estaVacio();
}
```


Interfaz Java para el iterador de árboles binarios

```
package adts;
public interface IterArbolBin<E> {
    // operaciones de modificación
    void insertaHijoIzquierdo(E elemento)
       throws YaExiste,NoValido;
    void insertaHijoDerecho(E elemento)
       throws YaExiste,NoValido;
    E eliminaHoja() throws NoEsHoja, NoValido;
    E modificaElemento (E elementoNuevo)
       throws NoValido;
    ArbolBinario<E> reemplazaRamaIzquierda
       (ArbolBinario<E> nuevaRama) throws NoValido;
    ArbolBinario<E> reemplazaRamaDerecha
       (ArbolBinario<E> nuevaRama) throws NoValido;
    // operaciones de consulta
```

Interfaz Java para el iterador de árboles binarios (cont.)

```
    E contenido() throws NoValido;
    boolean esHoja() throws NoValido;
    boolean esRaiz() throws NoValido;
    boolean tieneHijoIzquierdo() throws NoValido;
    boolean tieneHijoDerecho() throws NoValido;
    boolean esValido();
    // operaciones de recorrido
    void irARaiz();
    void irAHijoIzquierdo() throws NoValido;
    void irAHijoDerecho() throws NoValido;
    void irAPadre() throws NoValido;
        //duplicar un iterador
    IterArbolBin<E> clone();
    ArbolBinario<E> clonarArbol();
}
```


3.5. Búsquedas en árboles binarios

Los árboles binarios se adaptan muy bien para buscar elementos de forma eficiente.

Para ello, todos los elementos se almacenan en el árbol ordenados:

- **Todos los descendientes izquierdos de un nudo son menores que él**
- **Todos los descendientes derechos de un nudo son mayores que él**

En este caso, la búsqueda es O(log n) en el caso promedio, si el árbol está equilibrado

- **si las hojas están aproximadamente a la misma profundidad**

Existen algoritmos de inserción equilibrada (ej: *AVL, árboles roji-negros, ...*) que veremos en la sección de implementación

Algoritmo de inserción en un árbol binario ordenado

UNIVERSIDAD DE CANTABRIA

Insertar un elemento en un arbol binario ordenado, a partir del nudo indicado por el iterador. Si ya existe, no se inserta

```
método insertaOrdenado (E elem, IterArbolBin iter)
   si elem < iter.contenido()
      // vamos por la izquierda
      si iter tiene Hijo Izquierdo
             iter.irAHijoIzquierdo();
             insertaOrdenado(elem, iter);
      si no
             iter.insertaHijoIzquierdo(elem);
      fsi
```


Algoritmo de inserción en un árbol binario ordenado (cont.)

UNIVERSIDAD DE CANTABRIA

```
   si no, si elem > iter.contenido()
      // vamos por la derecha
      si iter tiene Hijo Derecho
         iter.irAHijoDerecho();
         insertaOrdenado(elem, iter);
      si no
         iter.insertaHijoDerecho(elem);
      fsi
   fsi
fmétodo
```

Codificación en Java

```java
public static <E extends Comparable<E>> void
     insertaOrdenado (E elem, IterArbolBin<E> iter)
{
   try {
      int comparacion=
         elem.compareTo(iter.contenido());
      if (comparacion<0) {
          // vamos por la izquierda
          if (iter.tieneHijoIzquierdo()) {
              iter.irAHijoIzquierdo();
              insertaOrdenado(elem, iter);
          } else {
              iter.insertaHijoIzquierdo(elem);
          }
```


Codificación en Java (cont.)

```java
      } else if (comparacion>0) {
          // vamos por la derecha
          if (iter.tieneHijoDerecho()) {
              iter.irAHijoDerecho();
              insertaOrdenado(elem, iter);
          } else {
              iter.insertaHijoDerecho(elem);
          }
      }
   } catch (YaExiste e) {
      System.out.println("Error inesperado: "+e);
   } catch (NoValido e) {
      System.out.println("Error inesperado: "+e);
   }
}
```

Algoritmo de búsqueda en un árbol binario ordenado

UNIVERSIDAD DE CANTABRIA

Buscar un elemento en un arbol binario ordenado, a partir del nudo indicado por el iterador, y retornando otro iterador cuyo nudo actual es el elemento encontrado, o `null` si no se encuentra

```
método buscaOrdenado (E elem, IterArbolBin<E> iter)
                      retorna IterArbolBin<E>
   si iter no es Valido
      // no encontrado
      retorna null;
   fsi
   si elem==iter.contenido
      // nudo encontrado
      retorna iter.clone();
```


Algoritmo de búsqueda en un árbol binario ordenado

UNIVERSIDAD DE CANTABRIA

```
   si no, si elem<iter.contenido
      // buscamos por la izquierda
      iter.irAHijoIzquierdo();
      retorna buscaOrdenado(elem, iter);
   si no
      // buscamos por la derecha
      iter.irAHijoDerecho();
      retorna buscaOrdenado(elem, iter);
   fsi
fmétodo
```

Codificación en Java

UC UNIVERSIDAD DE CANTABRIA

```
public <E extends Comparable<E>> IterArbolBin<E>
   buscaOrdenado (E elem, IterArbolBin<E> iter)
{
   if (!iter.esValido()) {
      // no encontrado
      return null;
   }
   try {
      int comparacion=
         elem.compareTo(iter.contenido());
      if (comparacion==0) {
         // nudo encontrado
         return iter.clone();
```


Codificación en Java (cont.)

UC UNIVERSIDAD DE CANTABRIA

```
      } else if (comparacion<0) {
         // buscamos por la izquierda
         iter.irAHijoIzquierdo();
         return buscaOrdenado(elem, iter);
      } else {
         // buscamos por la derecha
         iter.irAHijoDerecho();
         return buscaOrdenado(elem, iter);
      }
   } catch (NoValido e) {
      System.out.println("Error inesperado: "+e);
      return null;
   }
}
```