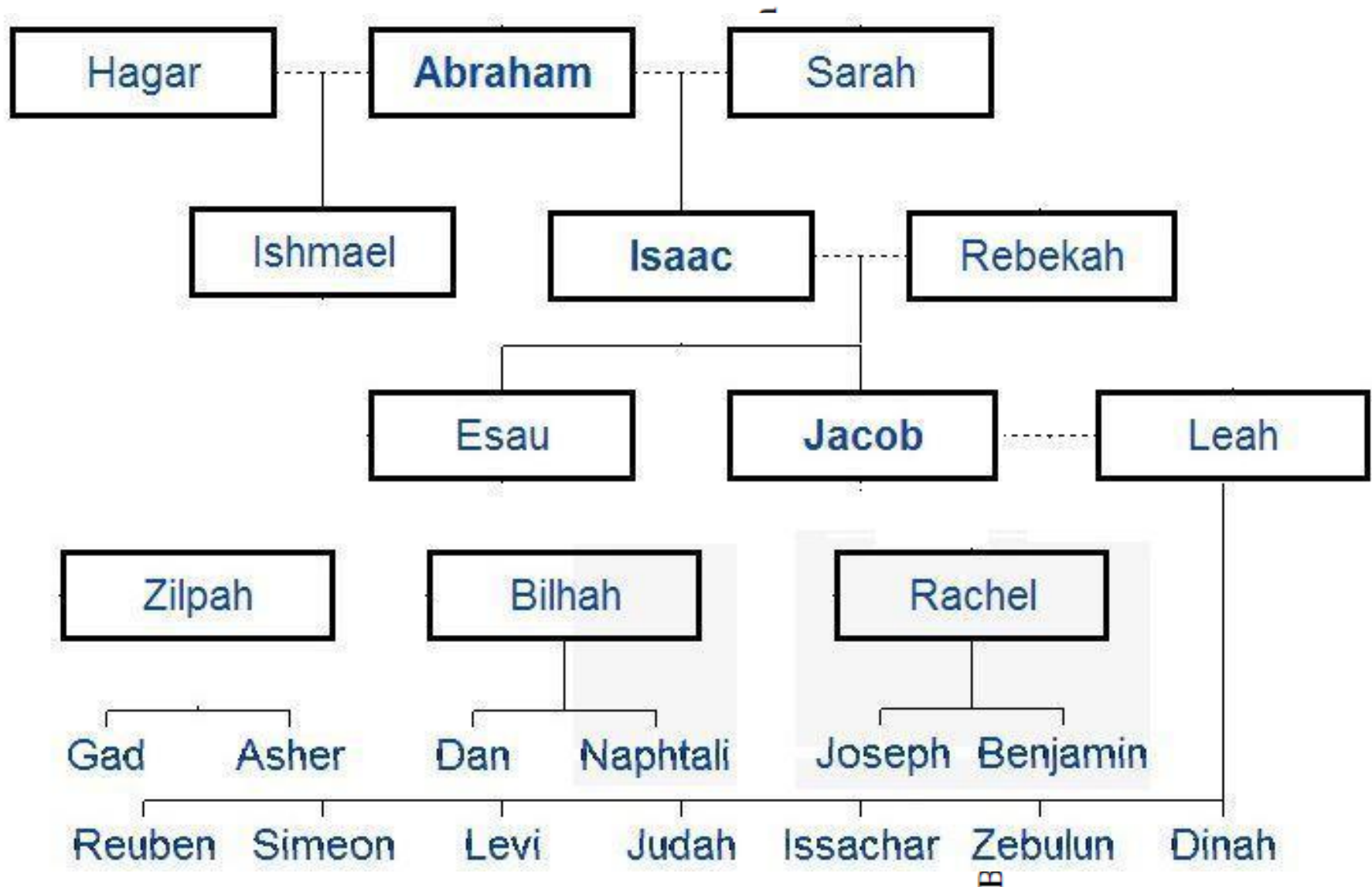


Arboles

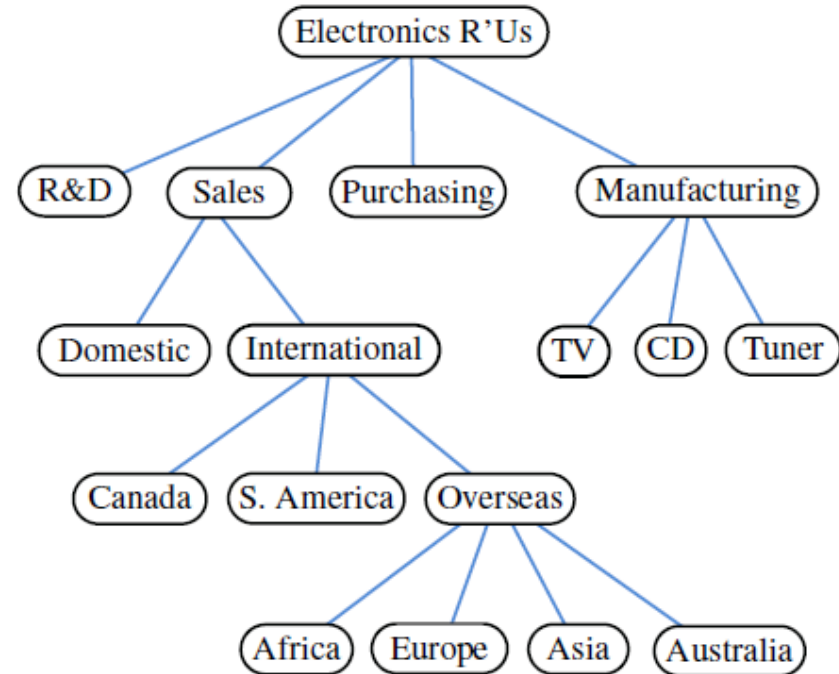
Algorítmica y Programación II

Algunos descendientes de Abraham



¿Qué es un árbol?

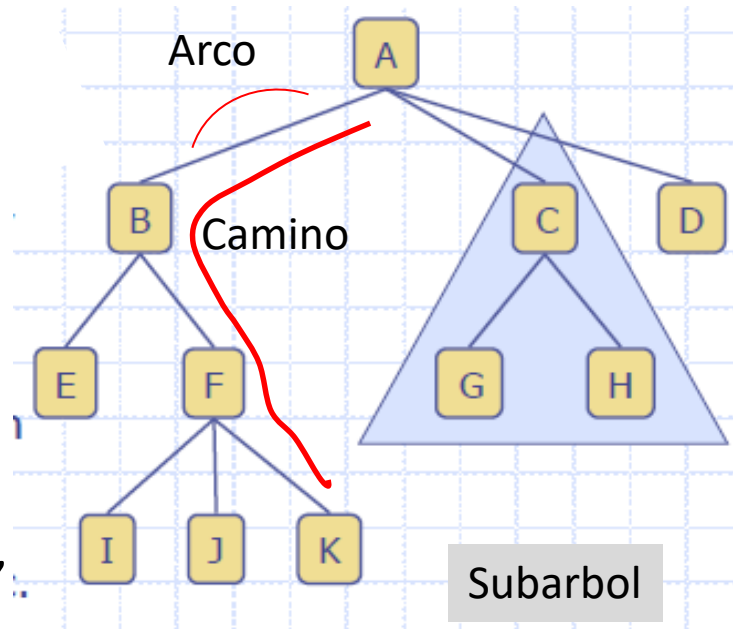
- Un **árbol** T es un conjunto de **nodos** que almacenan elementos tal que los nodos tienen una relación **padre-hijo** que satisface las siguientes propiedades:
 - Si T no es vacío, este tiene un nodo especial llamado **raíz** de T , este no tiene padre.
 - Cada nodo v de T diferente del **raíz** tiene un único nodo **padre** w ; cada nodo con padre w es un **hijo** de w .
- En informática, **un árbol es un modelo abstracto de una estructura jerárquica**
- Aplicaciones:
 - Gráficos organizacionales
 - Sistemas de archivos
 - Entornos de programación



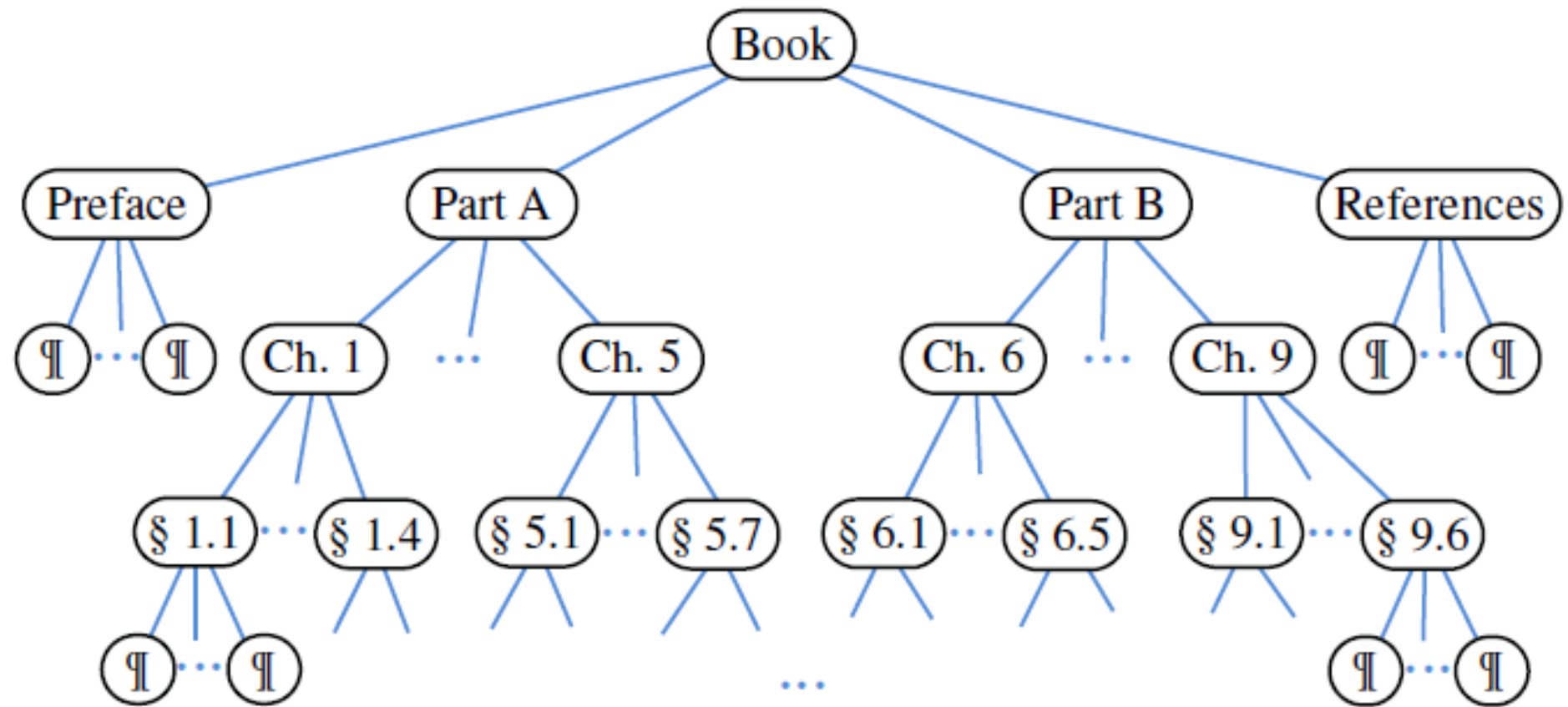
Terminología de árboles

- **Raíz:** nodo sin padre (A)
- **Nodo interno:** nodo con al menos un hijo (A, B, C, F)
- **Nodo externo (hoja):** nodo sin hijos (E, I, J, K, G, H, D)
- **Ancestros de un nodo:** padre, abuelo, bisabuelos, etc.
- **Profundidad de un nodo:** número de ancestros
- **Peso de un árbol:** máxima profundidad de nodos (3)
- **Descendientes de un nodo:** hijo, nieto, bisnieto, etc.
- **Subarbol:** árbol consistente de un nodo y sus descendientes

Diremos **arco** (*edge*) del **árbol T** dado un par de nodos (**u, v**) tales que **u** es el padre de **v**, o viceversa, que los une. Y **camino** (*path*) de **T** es una secuencia de nodos tal que cualesquiera dos nodos consecutivos en la secuencia forman un **arco**.



Arbol ordenado (Ej. Libro)



TAD árbol

Definimos un TAD árbol utilizando el concepto de ***posición*** como una abstracción para un nodo de un árbol. Se almacena un elemento en cada posición y las posiciones satisfacen las relaciones padre-hijo que definen la estructura del árbol. Un objeto ***posición*** (position) para un árbol soporta los métodos:

- **getElement()** retorna el elemento almacenado en la posición.
- **Métodos genéricos:**
 - integer size()
 - boolean isEmpty()
 - Iterator iterator()
 - Iterable positions()
- **Métodos de acceso:**
 - position root()
 - position parent(p)
 - Iterable children(p)
 - Integer numChildren(p)
- **Métodos de consulta:**
 - boolean isInternal(p)
 - boolean isExternal(p)
 - boolean isRoot(p)
- **Métodos de actualización**
adicionales se pueden definir para las estructuras de datos que implementan el TAD árbol

Interfaz JAVA

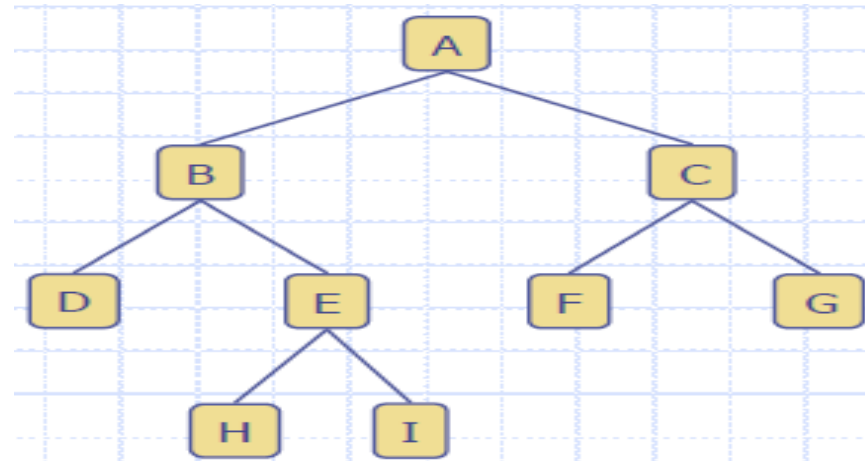
```
1  /** An interface for a tree where nodes can have an arbitrary number of children. */
2  public interface Tree<E> extends Iterable<E> {
3      Position<E> root();
4      Position<E> parent(Position<E> p) throws IllegalArgumentException;
5      Iterable<Position<E>> children(Position<E> p)
6          throws IllegalArgumentException;
7      int numChildren(Position<E> p) throws IllegalArgumentException;
8      boolean isInternal(Position<E> p) throws IllegalArgumentException;
9      boolean isExternal(Position<E> p) throws IllegalArgumentException;
10     boolean isRoot(Position<E> p) throws IllegalArgumentException;
11     int size();
12     boolean isEmpty();
13     Iterator<E> iterator();
14     Iterable<Position<E>> positions();
15 }
```

Cálculo de profundidad y altura de un árbol

- Sea **p** una posición dentro del árbol **T**. La **profundidad** de **p** es el número de **antepasados** de **p**, distintos de **p**.
 - Si **p** es la **raíz**, entonces la **profundidad** de **p** es 0.
 - De lo contrario, la **profundidad** de **p** es **uno más la profundidad del padre de p**.
- Y, definimos la **altura** de un árbol como igual al **máximo de las profundidades** de sus posiciones (o cero, si el árbol está vacío).
- **Nota:** ver códigos pág. 314, 315 y 316.

Arboles binarios

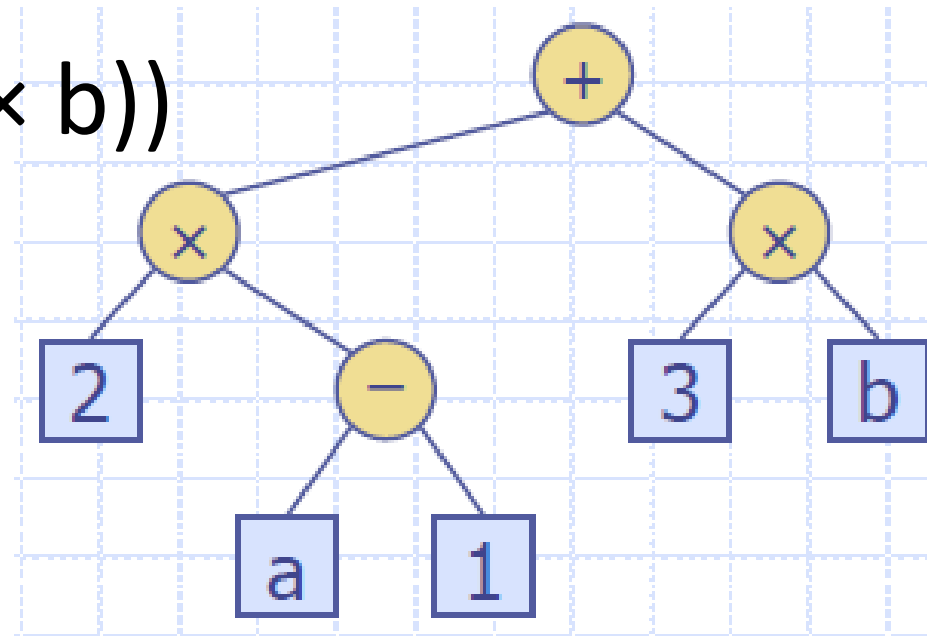
- **Propiedades:**
 - Cada nodo interno tiene como máximo dos hijos
 - Los hijos de un nodo son de orden par
- Llamamos a los hijos de un nodo interno ***Hijo-izquierdo*** e ***Hijo-derecho***
- Aplicaciones:
 - Expresiones aritméticas
 - Procesos de decisión
 - búsquedas



Árbol de expresiones aritméticas

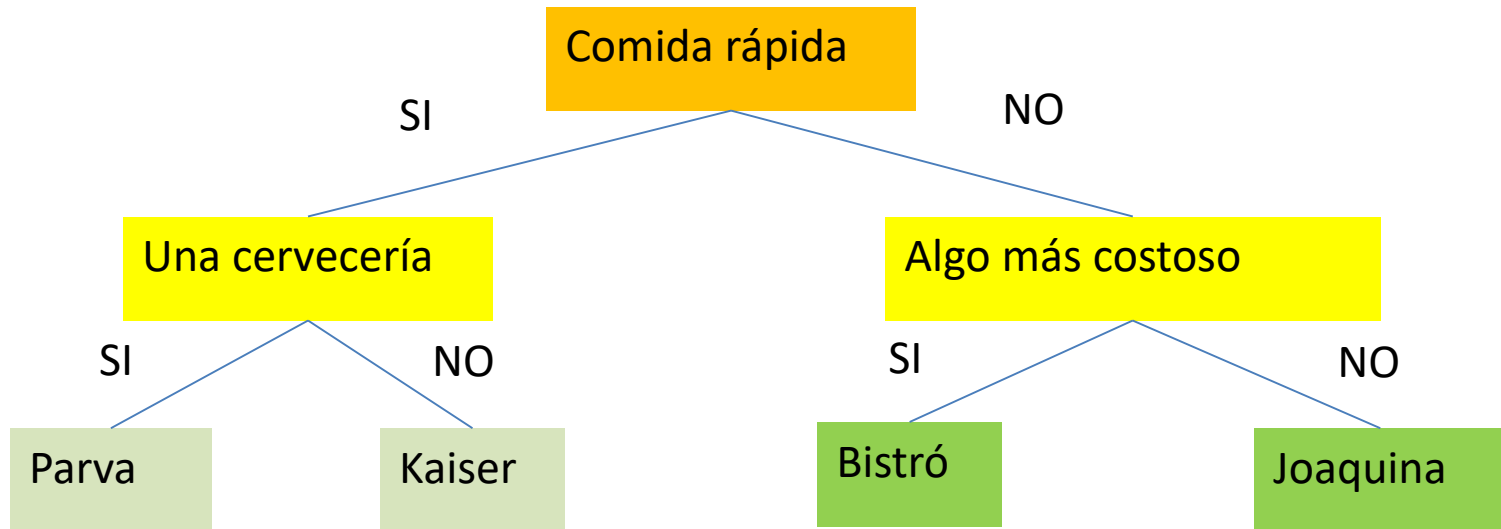
- **Nodos internos:** operadores
- **Nodo externos:** operandos
- Ejemplo: árbol de expresión aritmética para

$$(2 \times (a - 1) + (3 \times b))$$



Arbol de decisión

- **Nodos internos:** preguntas con respuesta si/no
- **Nodos externos:** decisiones
- Ejemplo: decisión de salir a comer



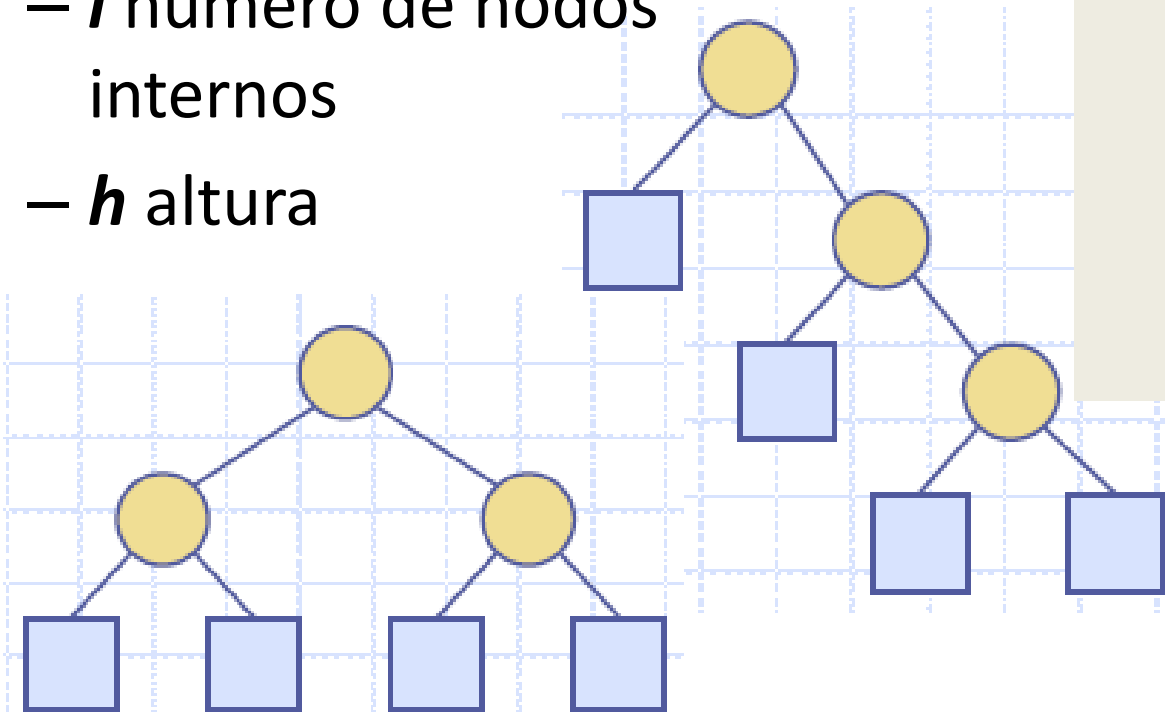
Propiedades de los árboles binarios

- Notación:

- n número de nodos
- e número de nodos externos
- i número de nodos internos
- h altura

- Propiedades:


- $e = i + 1$
- $n = 2e - 1$
- $h \leq i$
- $h \leq (n - 1)/2$
- $e \leq 2h$
- $h \geq \log_2 e$
- $h \geq \log_2 (n + 1) - 1$



Definición recursiva de un árbol binario

- un **árbol binario** es:
 - Un **árbol vacío**.
 - Un árbol **no vacío** que tiene un nodo raíz **r**, que almacena un elemento y dos árboles binarios que son respectivamente los **subárboles izquierdo** y **derecho** de **r**.
- **Nota:** uno o ambos de esos subárboles pueden estar vacíos según esta definición.

El TAD árbol binario

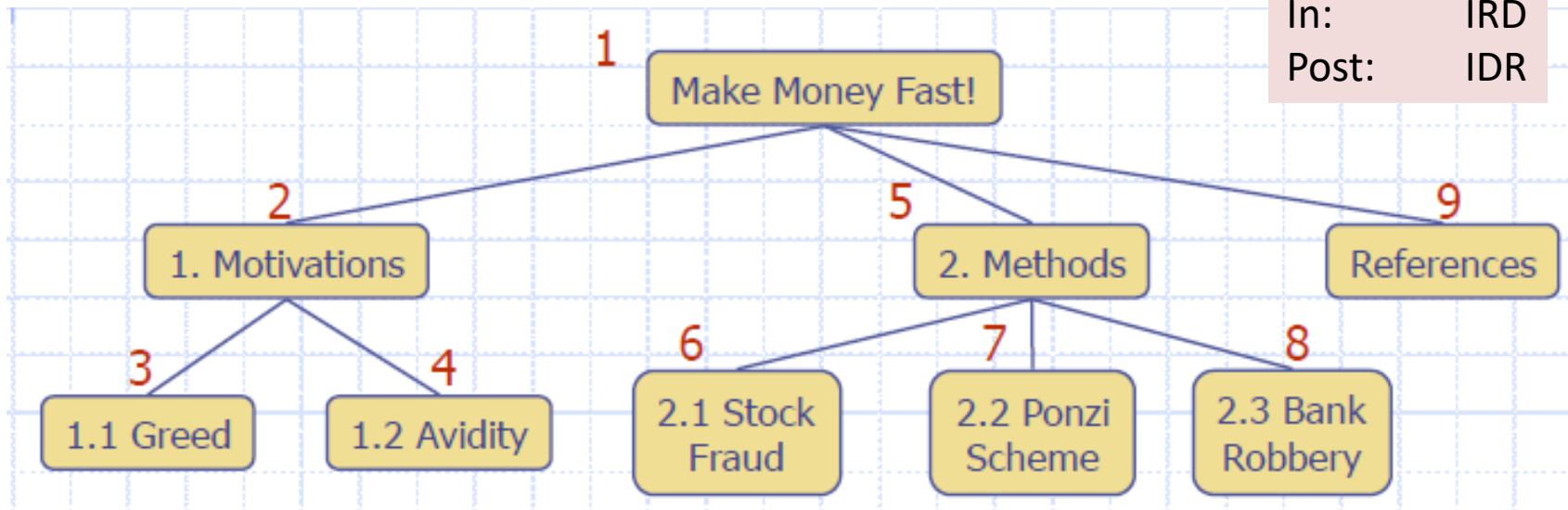
- Hereda del TAD árbol
 - Método adicionales:
 - **position left(p)**
 - **position right(p)**
 - **position sibling(p)**
 - Los métodos anteriores devuelven *null* cuando no hay a hijo izquierdo, derecho o hermano de de p , respectivamente
 - Los métodos de actualización se pueden definir por estructuras de datos que implementan el TAD árbol binario
-  Hermano
- Ver códigos pag. 319-320

Recorrido PreOrden

- Un recorrido visita de manera sistemática los nodos de un árbol
- En un recorrido preOrden, un nodo se visita antes que sus descendientes
- Aplicación: imprimir un documento estructurado

```
Algorithm preOrder(v)  
  visit(v)  
  for each child w of v  
    preorder (w)
```

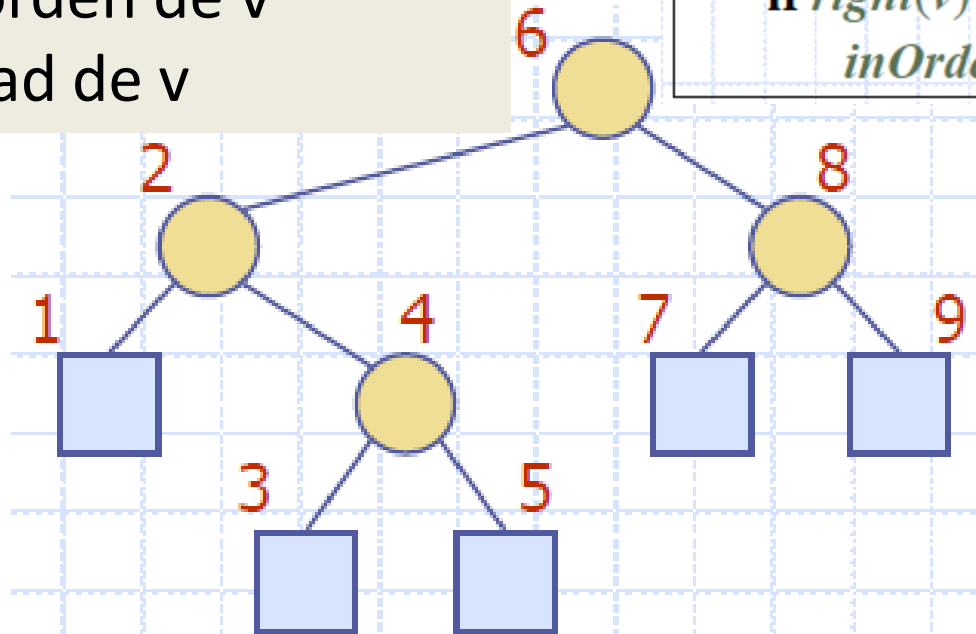
Pre:	RID
In:	IRD
Post:	IDR



Recorrido Enorden

- Un recorrido enOrden (inorder traversal) un nodo se visita luego del subárbol izquierdo y antes que el subárbol derecho
- Aplicación: Dibujar un árbol binario
 - $x(v)$ = rango de orden de v
 - $y(v)$ = profundidad de v

```
Algorithm inOrder( $v$ )  
  if  $\text{left}(v) \neq \text{null}$   
    inOrder( $\text{left}(v)$ )  
  visit( $v$ )  
  if  $\text{right}(v) \neq \text{null}$   
    inOrder( $\text{right}(v)$ )
```



Impresión de expresiones aritméticas

- Especialización del recorrido enOrden
 - Imprimir operando u operador cuando se visita el nodo
 - Imprimir “(“ antes de recorrer el subárbol izquierdo
 - Imprimir “)” luego de recorrer el subárbol derecho

Algorithm *printExpression(v)*

if *left(v) ≠ null*

print (“(”)

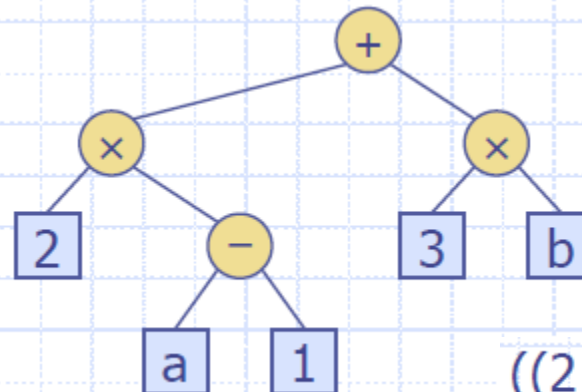
inOrder (*left(v)*)

print (*v.element* ())

if *right(v) ≠ null*

inOrder (*right(v)*)

print (“)”)

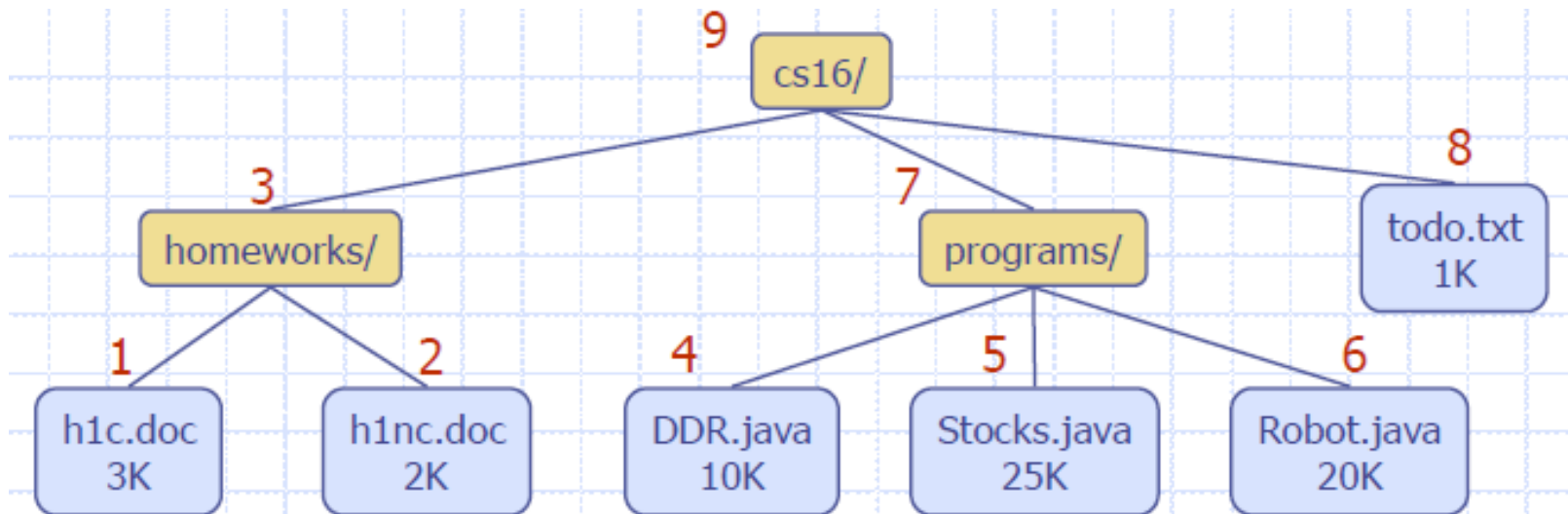


$((2 \times (a - 1)) + (3 \times b))$

Recorrido PostOrden

- Un nodo se visita luego de sus descendientes
- Aplicación: El espacio utilizado en una computadora por archivos, carpetas y subcarpetas

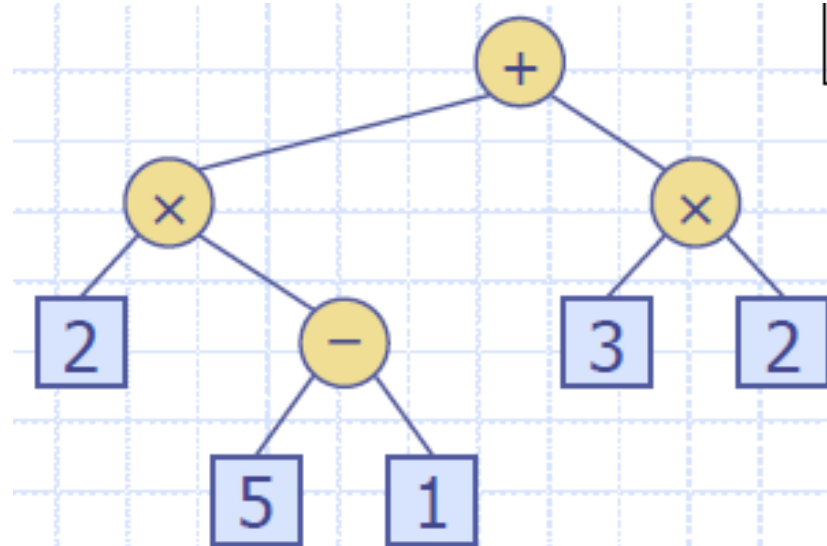
```
Algorithm postOrder(v)
  for each child w of v
    postOrder(w)
  visit(v)
```



Evaluación de expresiones aritméticas

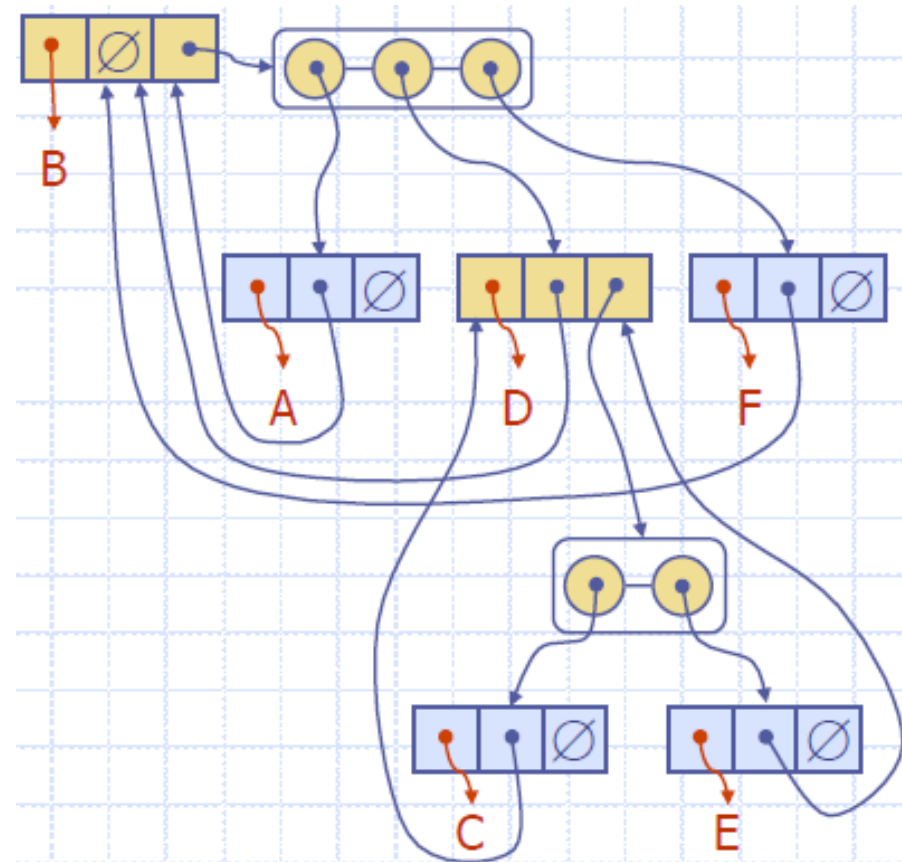
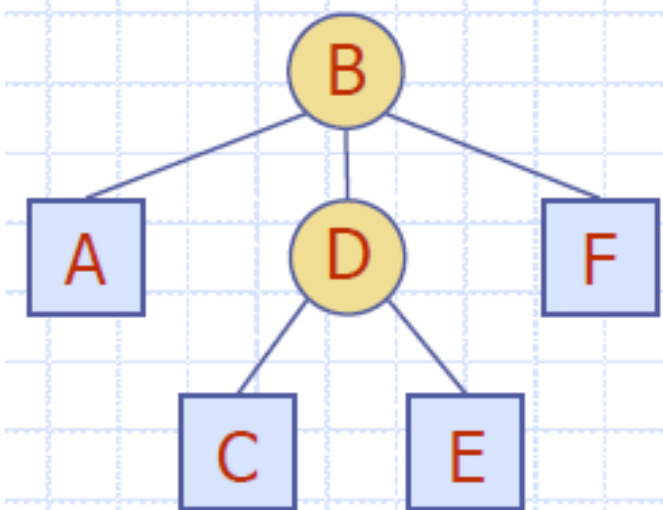
- Especialización de un recorrido postOrden
- El método recursivo retorna el valor de un subárbol
- Cuando visita un nodo interno, combina los valores de los subárboles

```
Algorithm evalExpr(v)  
  if isExternal(v)  
    return v.element()  
  else  
     $x \leftarrow evalExpr(left(v))$   
     $y \leftarrow evalExpr(right(v))$   
     $\diamond \leftarrow$  operator stored at v  
    return  $x \diamond y$ 
```



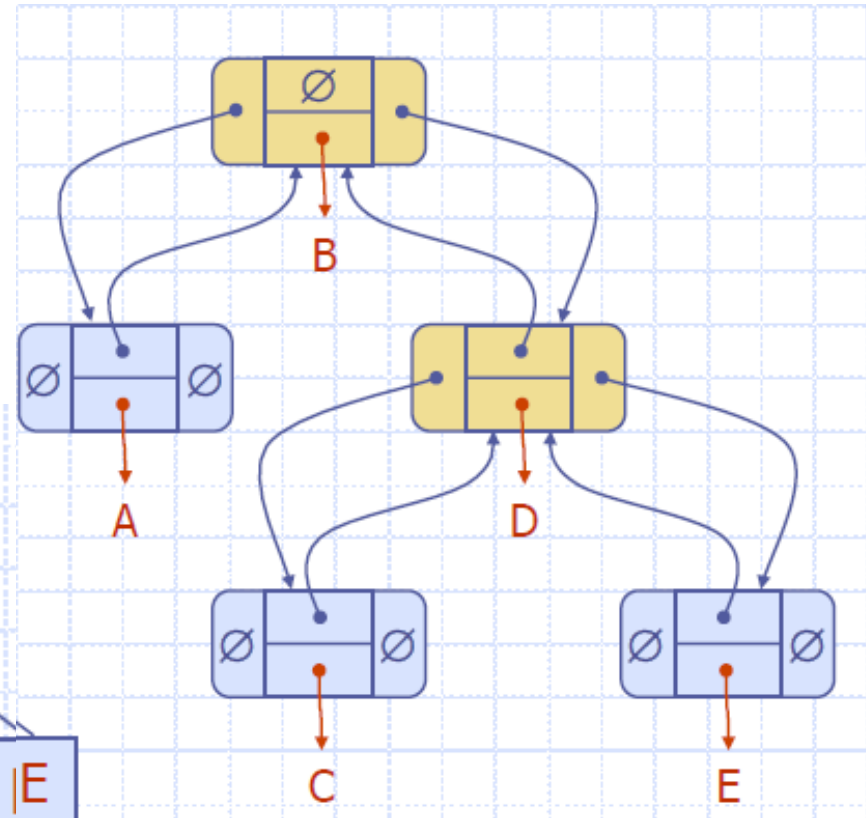
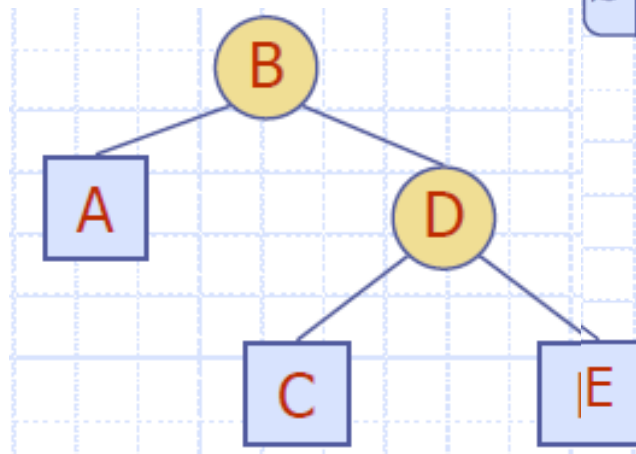
Implementación: Estructura enlazada

- Un nodo se representa por un objeto que almacena:
 - Elemento
 - Nodo padre
 - Secuencia de nodos hijos
- Los objetos de nodo implementan el TAD posición



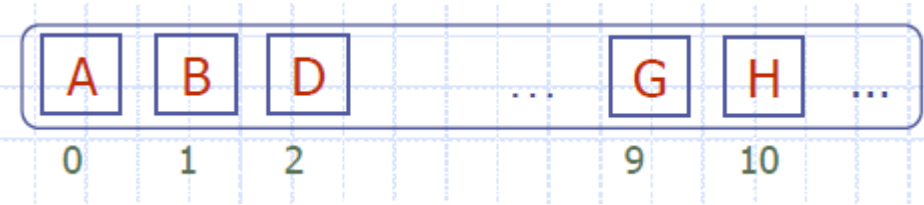
Estructura enlazada para árboles binarios

- Un nodo se presenta por un objeto que almacena:
 - Elemento
 - Nodo padre
 - Nodo hijo izquierdo
 - Nodo hijo derecho
- Los objetos nodo implementan el TAD Position



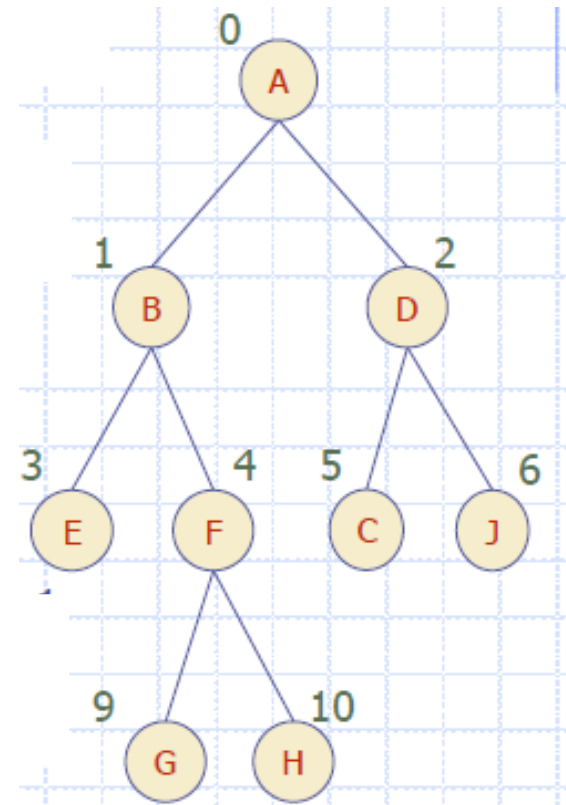
Representación de árboles binarios con arreglos

- Los nodos se almacenan en un arreglo A

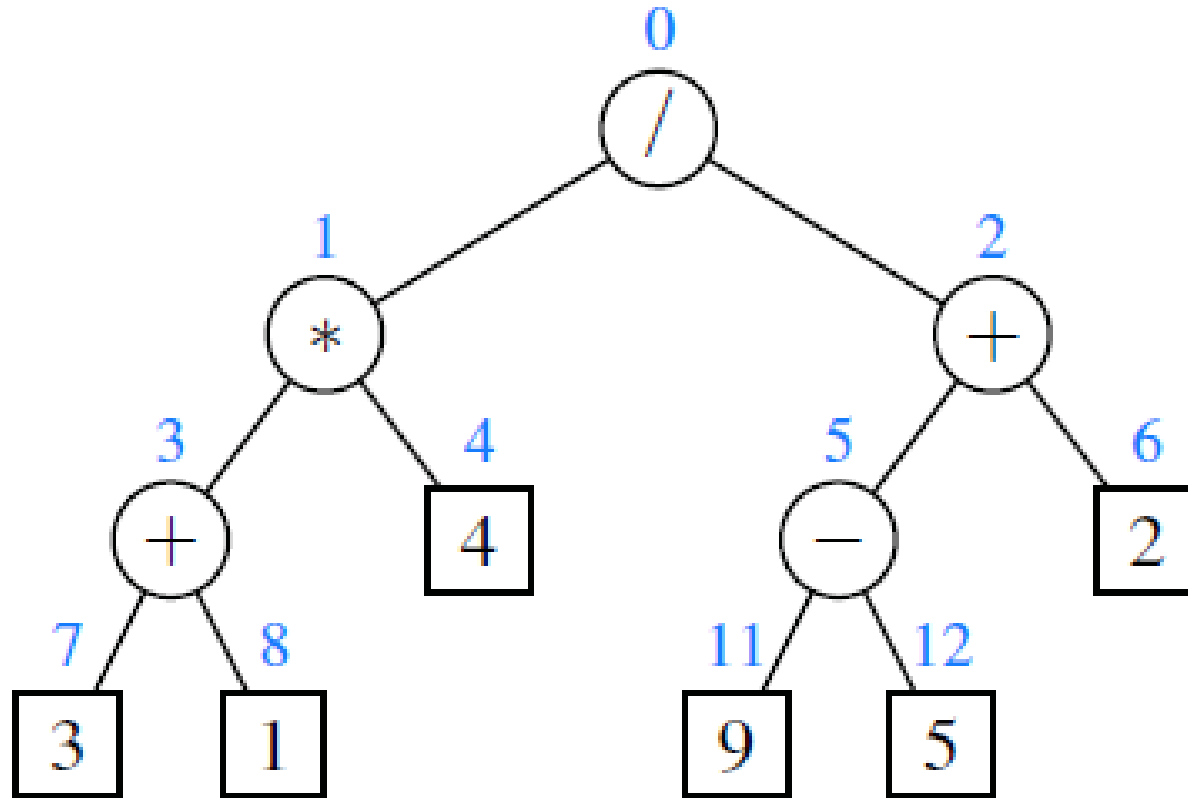


El nodo v se almacena en $A[\text{rank}(v)]$

- $\text{rank}(\text{root}) = 0$
- Si el nodo es el hijo izquierdo del $\text{parent}(\text{node})$, $\text{rank}(\text{node}) = 2 \cdot \text{rank}(\text{parent}(\text{node})) + 1$
- Si el nodo es el hijo derecho del $\text{parent}(\text{node})$, $\text{rank}(\text{node}) = 2 \cdot \text{rank}(\text{parent}(\text{node})) + 2$



Implementación de árboles binarios con arreglos



/	*	+	+	4	-	2	3	1			9	5		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Recorrido tour de Euler

- Recorrido genérico de un árbol binario
- Incluye un caso especial de recorrido
- Camina recorriendo el árbol y visita cada nodo tres veces:

Algorithm eulerTour(T, p):

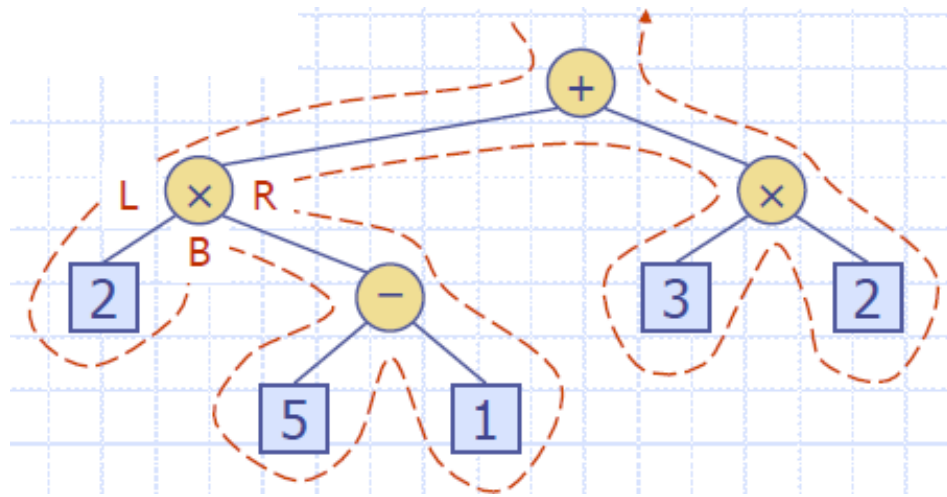
perform the “pre visit” action for position p

for each child c in $T.children(p)$ do

 eulerTour(T, c)

 { recursively tour the subtree rooted at c }

perform the “post visit” action for position p



Y seguimos viendo
códigos pág. 326 -
349

Bibliografía

- Data Structures and Algorithms in Java™. Sixth Edition. Michael T. Goodrich, Department of Computer Science University of California. Roberto Tamassia, Department of Computer Science Brown University. Michael H. Goldwasser, Department of Mathematics and Computer Science Saint Louis University. Wiley. 2014.