



UNIVERSITAT DE BARCELONA



Estructura de datos

Recorrido de Árboles Binarios

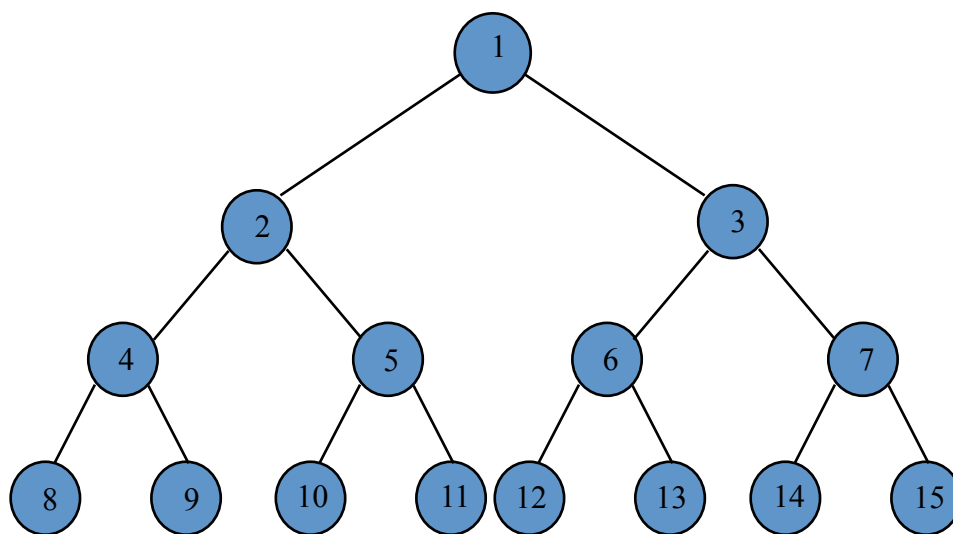
Santi Seguí | 2014-15

Índice

- ¿Qué es un Árbol?
- ¿Como definir una estructura tipo Árbol?
- ¿Como recorrer un Árbol?
 - Recorrido en anchura
 - Recorridos en profundidad
 - Inorder
 - Postorden
 - Preorden
 - Recorrido de Euler

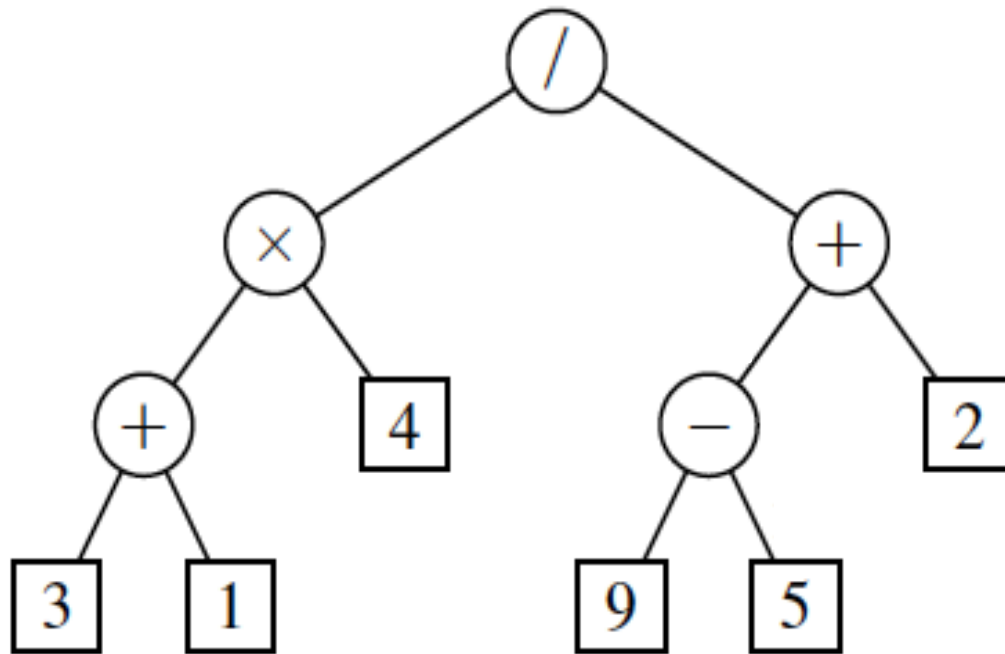
¿Que vimos en la última sesión?

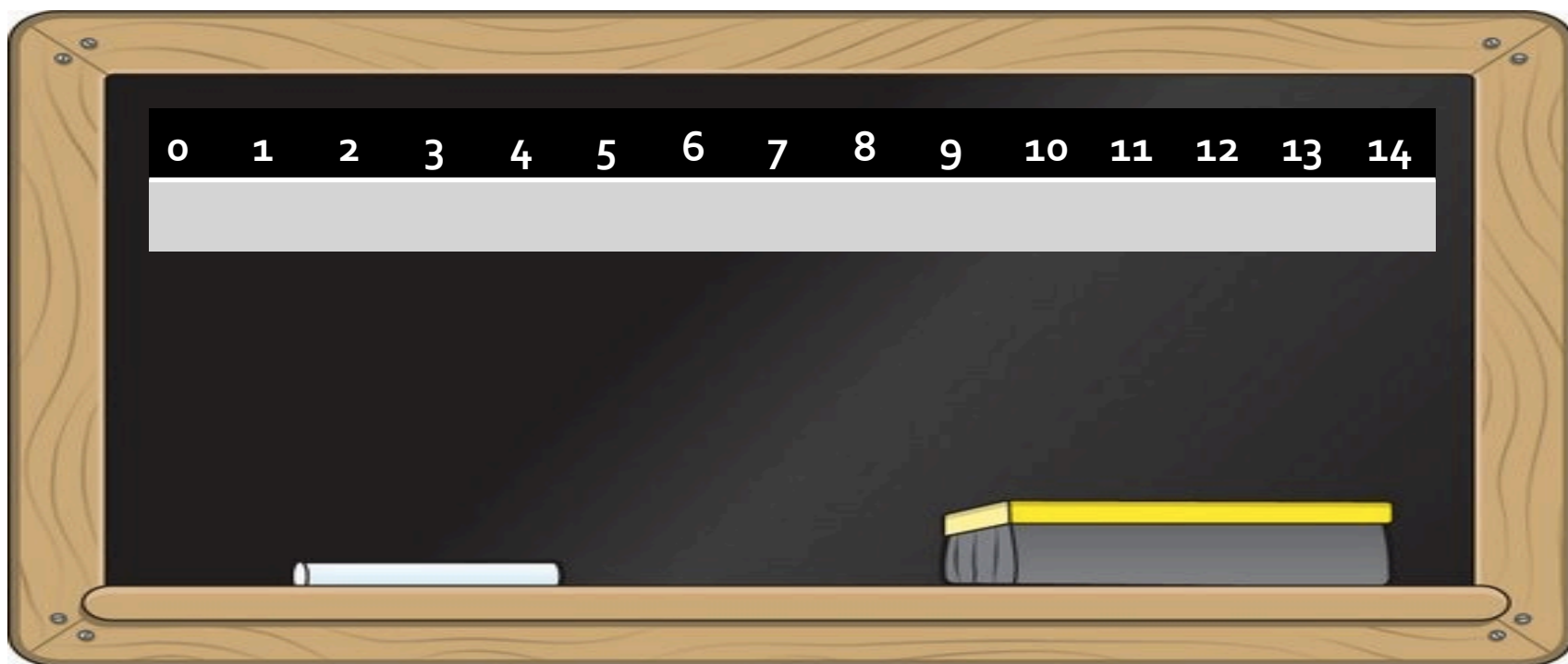
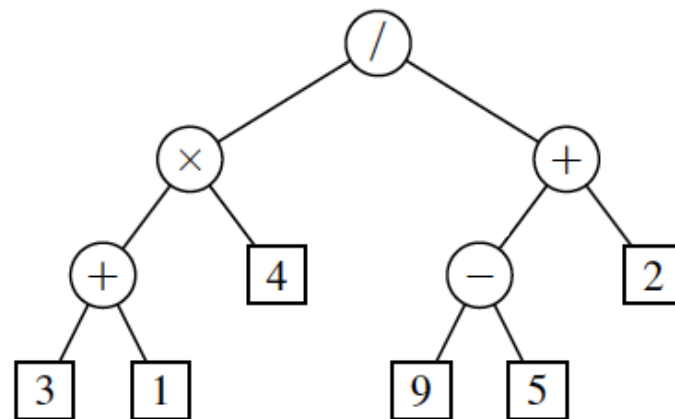
- Árboles & Árboles Binarios

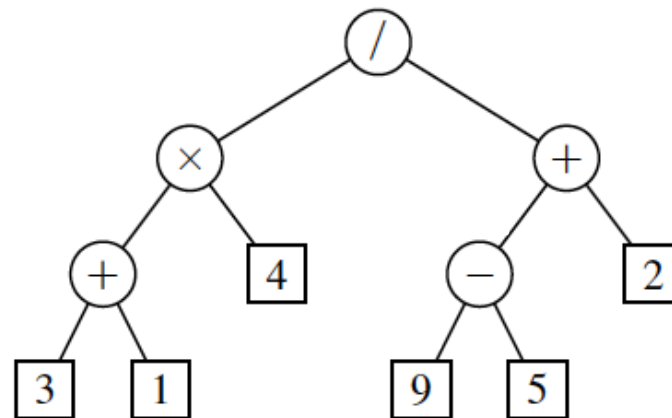


¿Cómo definir una estructura tipo Árbol?

- Representación Vectorial





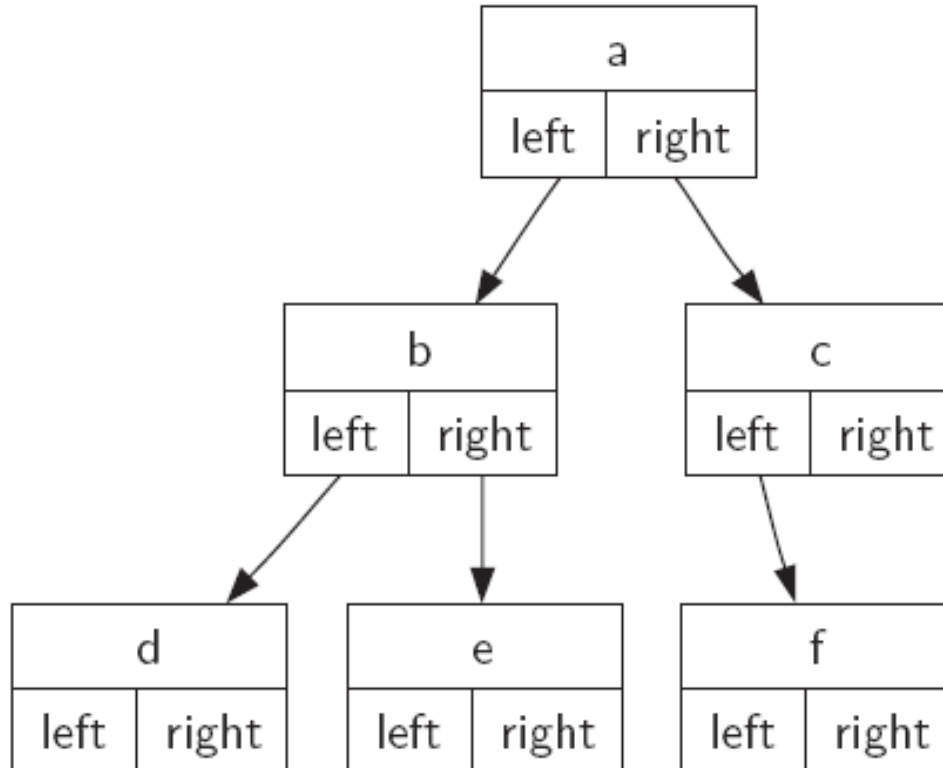


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
/	x	+	+	4	-	2	3	1			9	5		

El hijo izquierdo del nodo i se encuentra en la posición $2*i + 1$
El hijo derecho del nodo i se encuentra en la posición $2*i + 2$

¿Como definir una estructura tipo Arbol?

- Representación con estructuras enlazadas

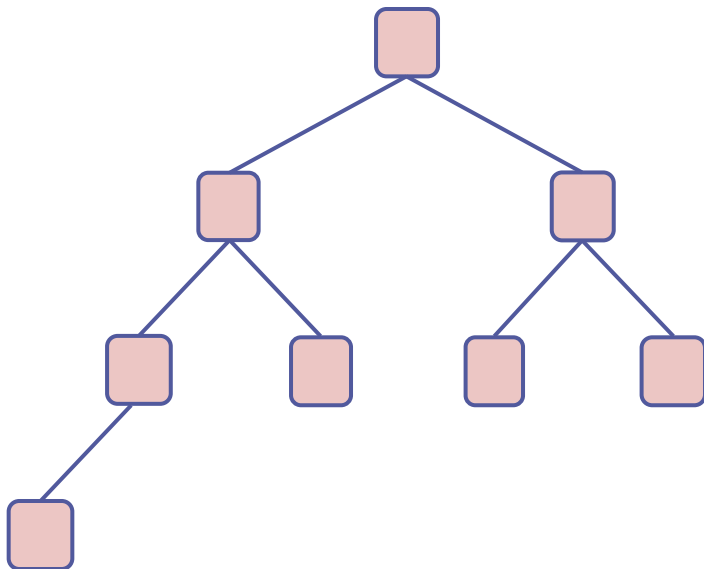


- **Un árbol binario es perfecto** si cada nivel esta completamente lleno.

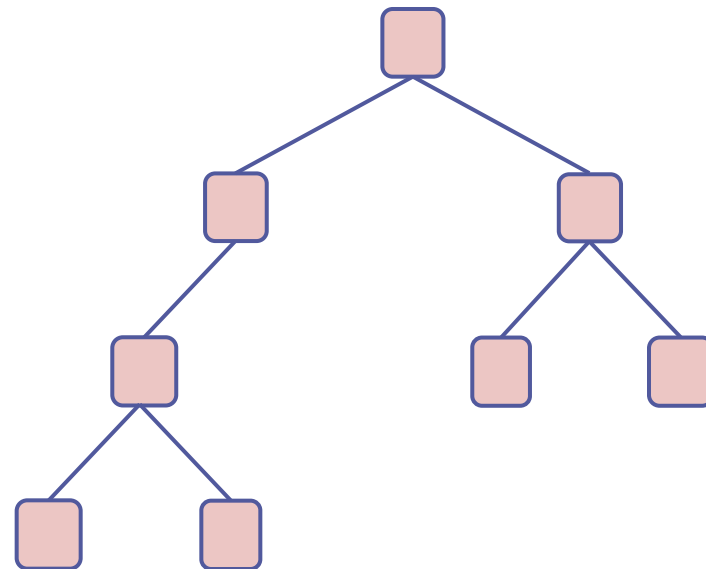


Completo

- Un árbol binario es casi-completo (o completo) si:
 - Cada nivel este completamente lleno, excluyendo el nivel mas bajo
 - Todos están lo máximo a la izquierda posible

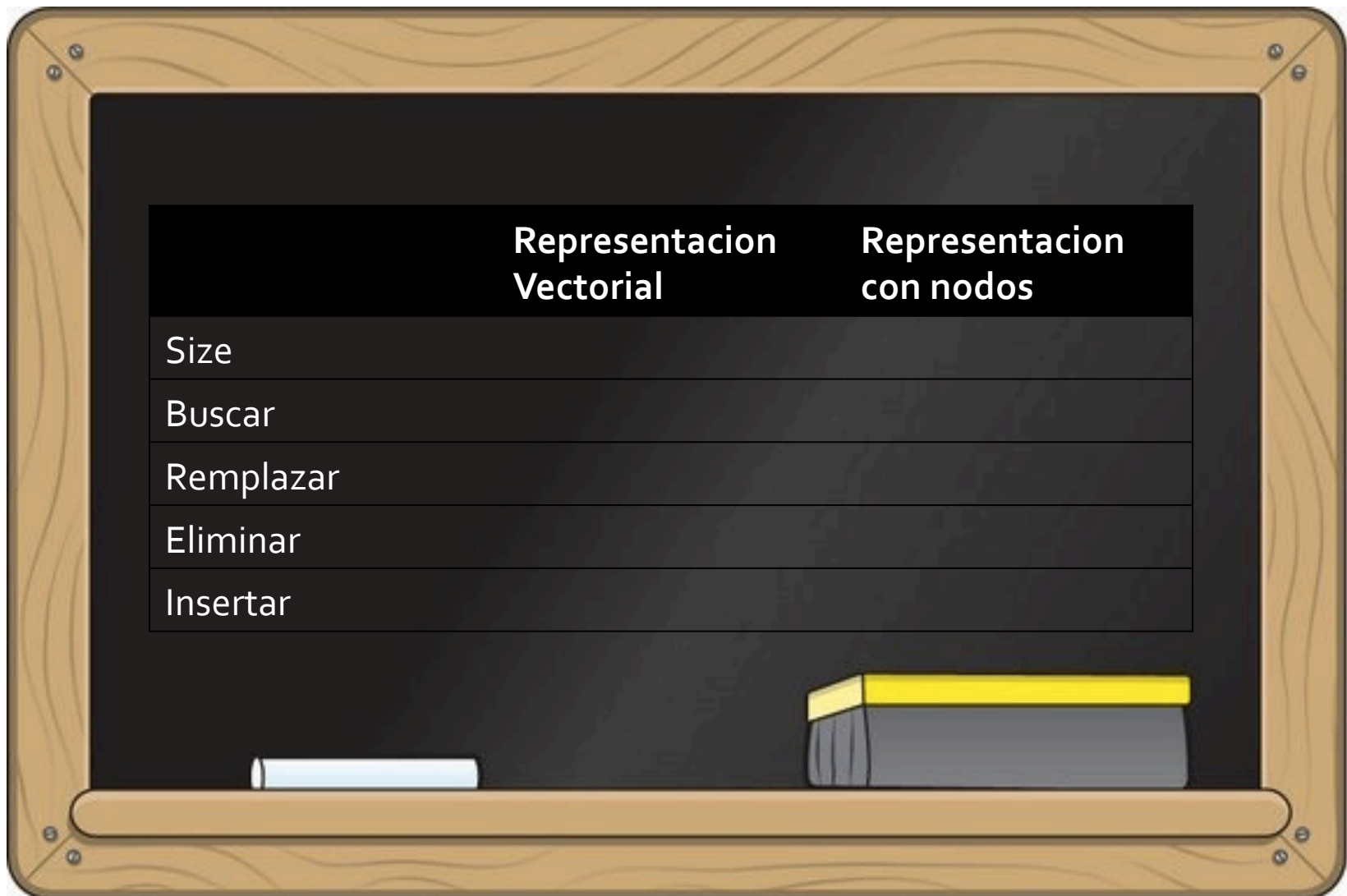


Casi-completo!



No completo ni casi-completo

Operaciones sobre arboles



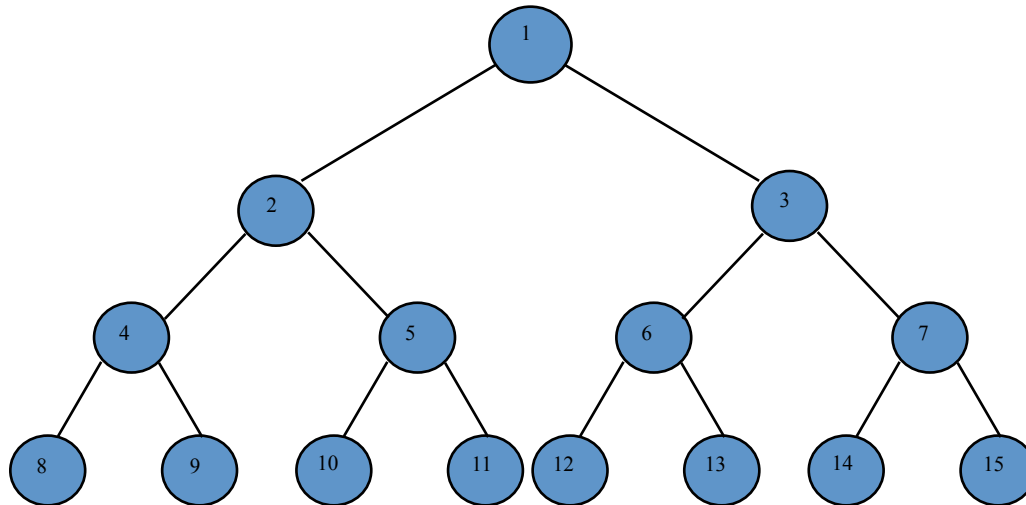
	Representacion Vectorial	Representacion con nodos
Size		
Buscar		
Reemplazar		
Eliminar		
Insertar		

Árbol como TDA

BinaryTree()	creates a new instance of a binary tree.
getLeft()	returns the binary tree corresponding to the left child of the current node.
getRight()	returns the binary tree corresponding to the right child of the current node.
setRootVal(val)	stores the object in parameter val in the current node.
getRootVal()	returns the object stored in the current node.
insertLeft(val)	creates a new binary tree and installs it as the left child of the current node.
insertRight(val)	creates a new binary tree and installs it as the right child of the current node.

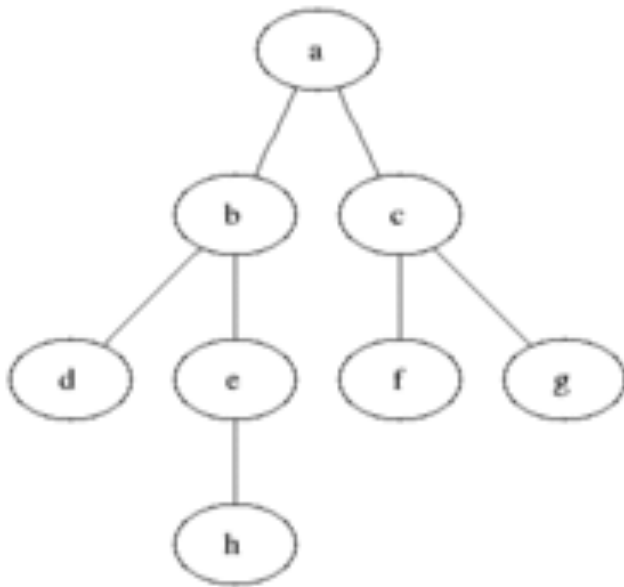
¿Como podemos recorrer un árbol binario?

- Que estrategias podemos seguir?
 - Top to Bottom? Left to Right?

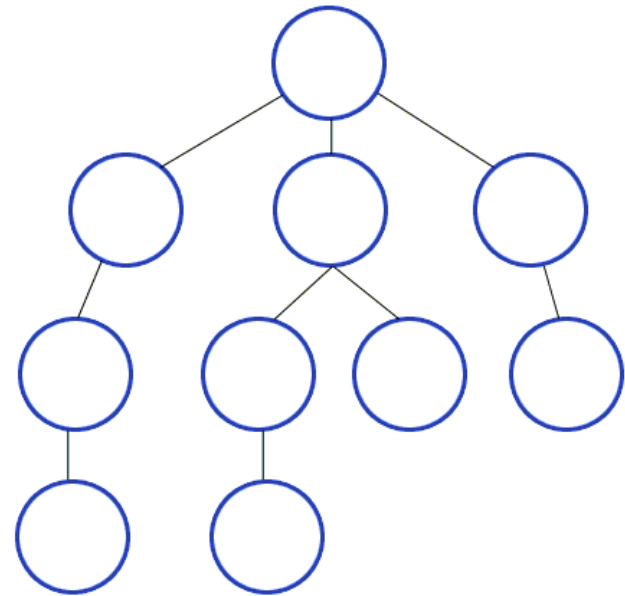


Recorrido Anchura vs. Profundidad

BFT



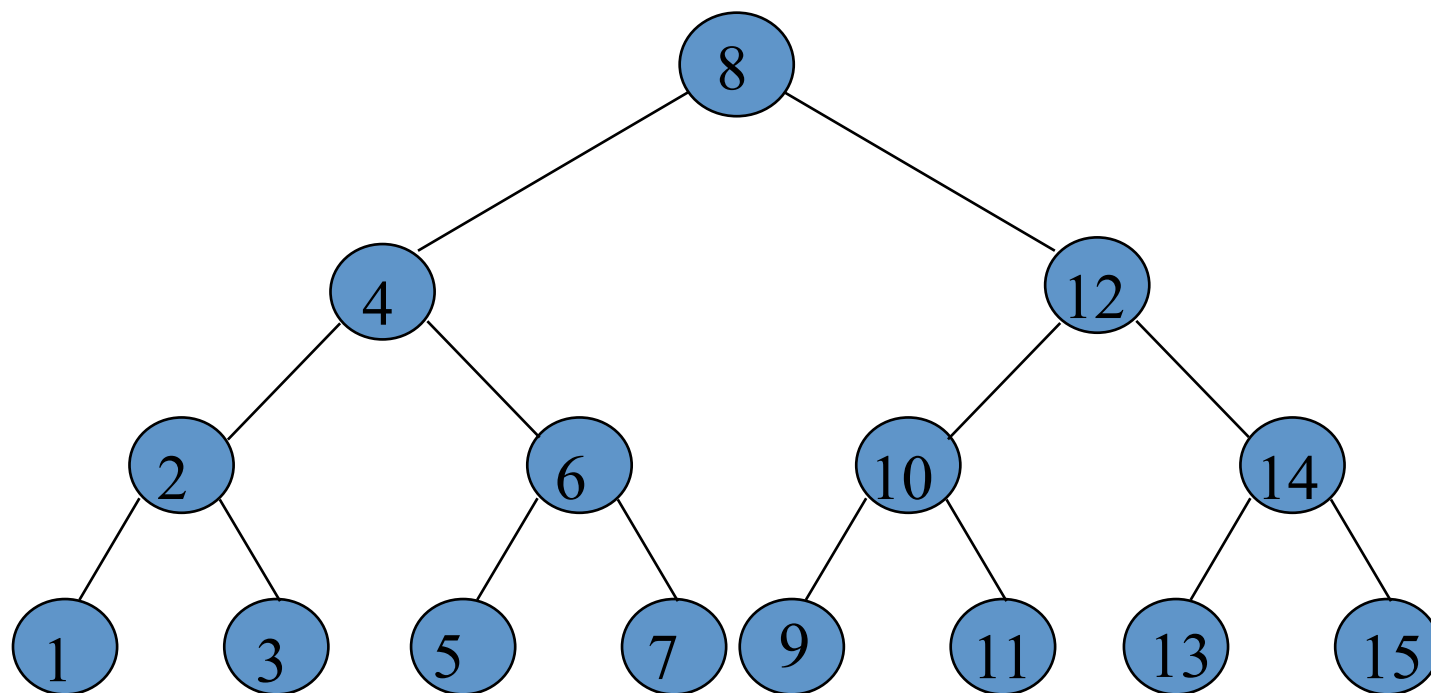
DFT



BFT gif: http://en.wikipedia.org/wiki/Breadth-first_search#mediaviewer/File:Animated_BFS.gif

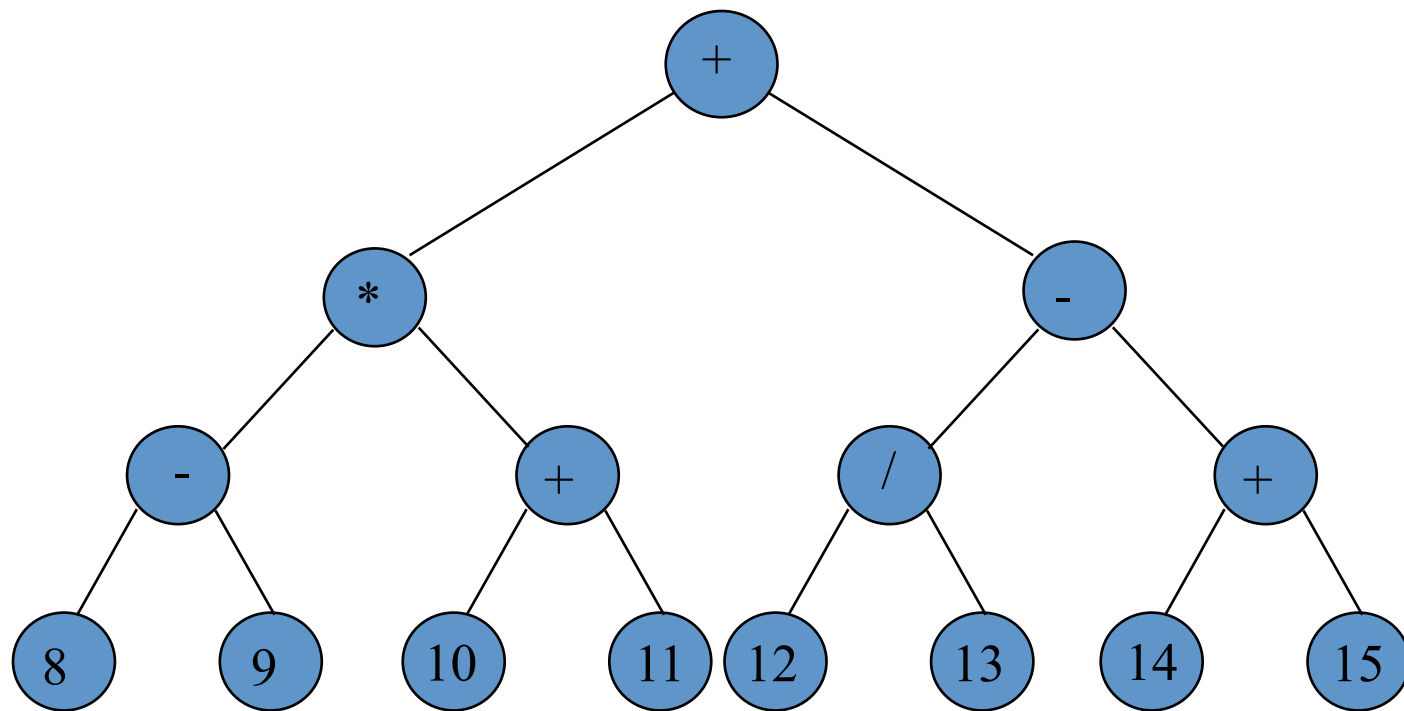
DFT gif: <https://upload.wikimedia.org/wikipedia/commons/7/7f/Depth-First-Search.gif>

¿Como podemos recorrer un árbol binario?



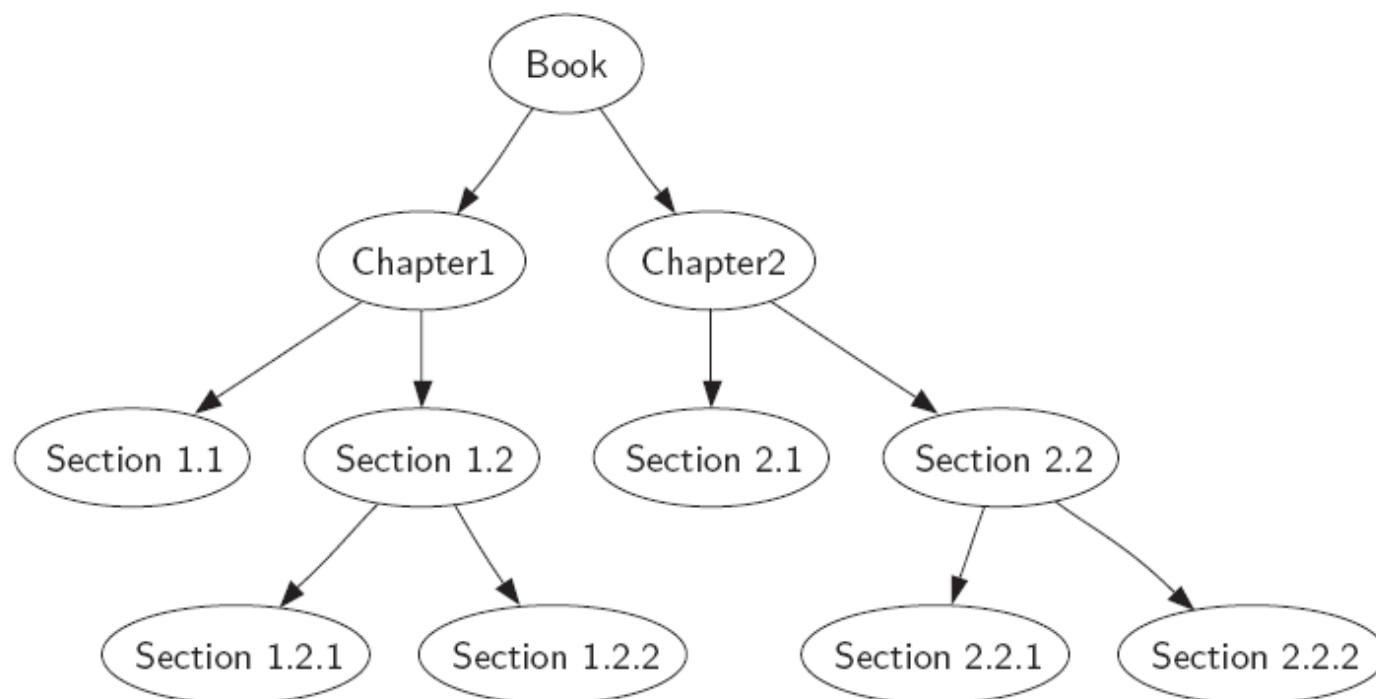
1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 - 13 - 14 - 15

¿Como podemos recorrer un árbol binario?



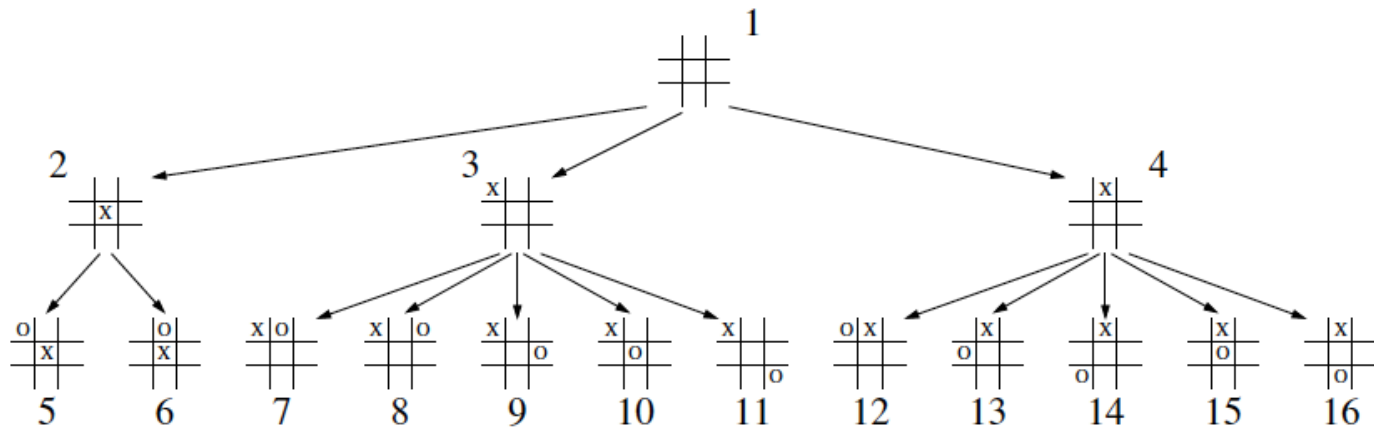
$((8 - 9) * (10 + 11)) + ((12/13) - (14+15))$

¿Como podemos recorrer un árbol binario?



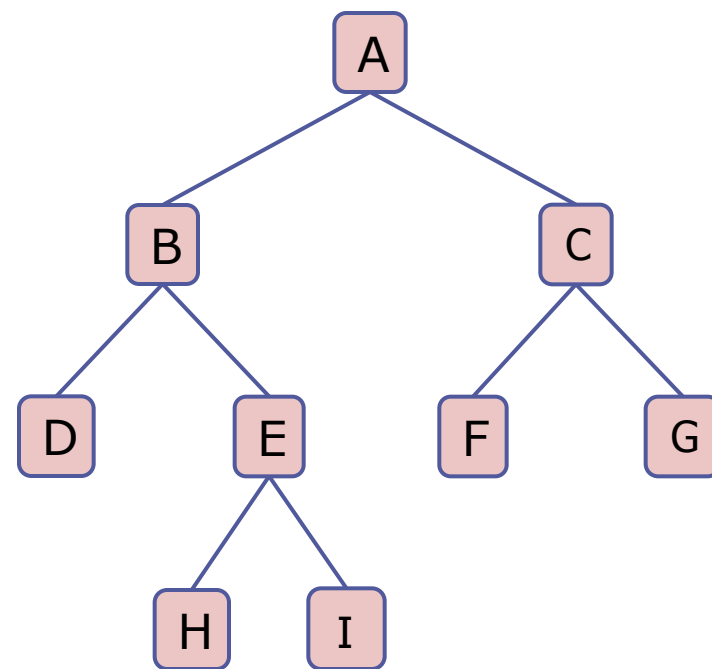
Recorrido por anchura

- Recorrido **breadth-first**
 - El recorrido en anchura es muy común en el desarrollo de juegos. Un árbol de juego representa los posibles movimientos en el juego que puede hacer un jugador o ordenador, siendo la raíz del árbol el movimiento inicial.



Recorrido por anchura: Breadth-first

- Empezamos desde el nodo raíz, visitamos todos sus hijos, después visitamos todos sus nietos, después sus bisnietos, después



**Recorrido
Breadth-first:**
A,B,C,D,E,F,G,H,I

Recorrido por anchura (2)

Estrategia general: Utilizamos una **cola** (queue) para hacer un seguimiento del orden de los nodos.

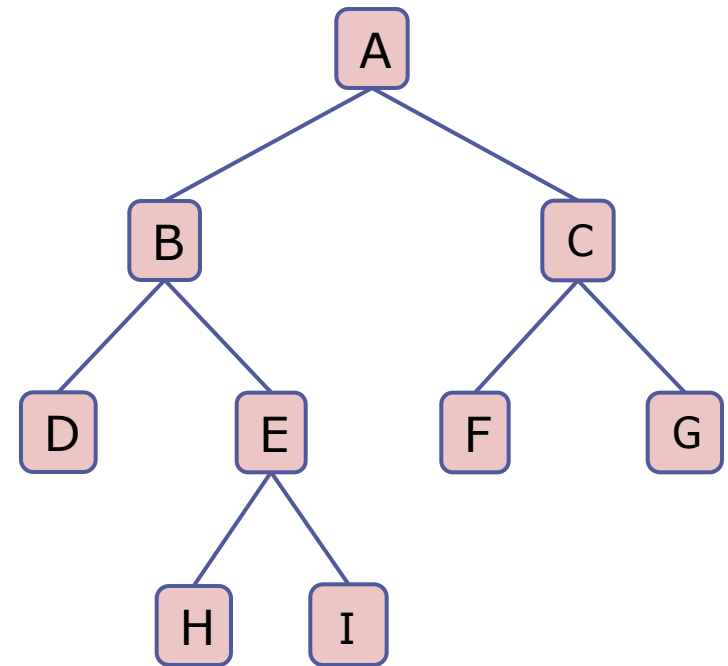
Pseudo-código:

```
function bft(root):  
    //Input: root node of tree  
    //Output: None  
    Q = new Queue()  
    enqueue root  
    while Q is not empty:  
        node = Q.dequeue()  
        visit(node)  
        enqueue node's left & right children
```

La cola nos garantiza que todos los nodos de un mismo nivel serán visitados antes que cualquier de sus hijos

Recorrido por profundidad

- Empezamos desde el nodo raíz, exploramos cada rama lo más profundamente posible antes de realizar backtracking
- Esto puede generar ordenaciones muy distintas dependiendo de que rama visitamos primero



**Recorrido en
profundidad:**

A,C,G,F,B,E,I,H,D

or

A,B,D,E,H,I,C,F,G

Recorrido por profundidad (2)

- Estrategia General: Utilizar una **pila** para hacer un seguimiento del orden de los nodos.
- Pseudo-código:

```
function dft(root):  
    //Input: root node of tree  
    //Output: none  
    S = new Stack()  
    push root onto S  
    while S is not empty:  
        node = S.pop()  
        visit(node)  
        push node's right and left children
```

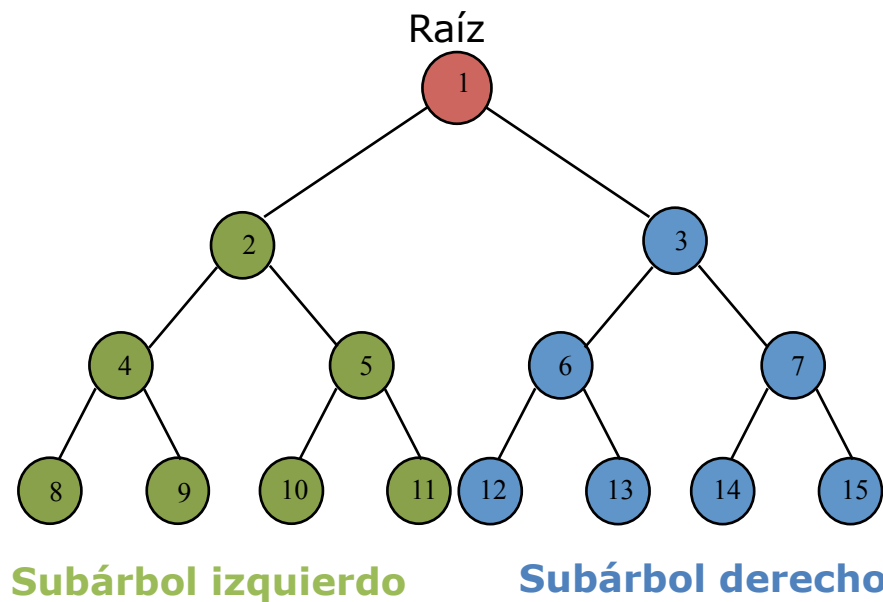
- La pila nos garantiza que exploraremos cada rama hasta llegar a la hoja antes de explorar otra.

¿Otras soluciones?

- Los métodos de **recorrido en profundidad** son especialmente óptimos usando la **recursividad**

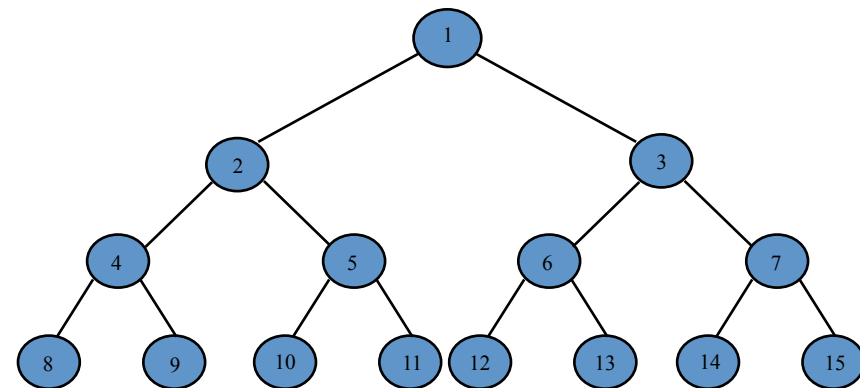
Recursividad en Árboles Binarios

- Planteamiento:
 - Caso general: se aplica la operación recursiva sobre:
 - el subárbol izquierdo, y
 - el subárbol derecho.
 - Caso particular: árbol vacío.



¿Cómo podemos recorrer un árbol binario por profundidad?

- Si denominamos con
 - L: movimiento a la izquierda
 - V: “visitar” el nodo
 - R: movimiento a la derecha



- Tenemos seis combinaciones posibles de recorrido:
 - **LVR**, **LRV**, **VLR**, VRL, RVL, RLV
- Si optamos por realizar primero el movimiento a izquierda, tenemos las tres primeras posibilidades
 - LVR lo llamaremos **inorden**,
 - VLR lo llamaremos **preorden**, y
 - LRV lo llamaremos **postorden**

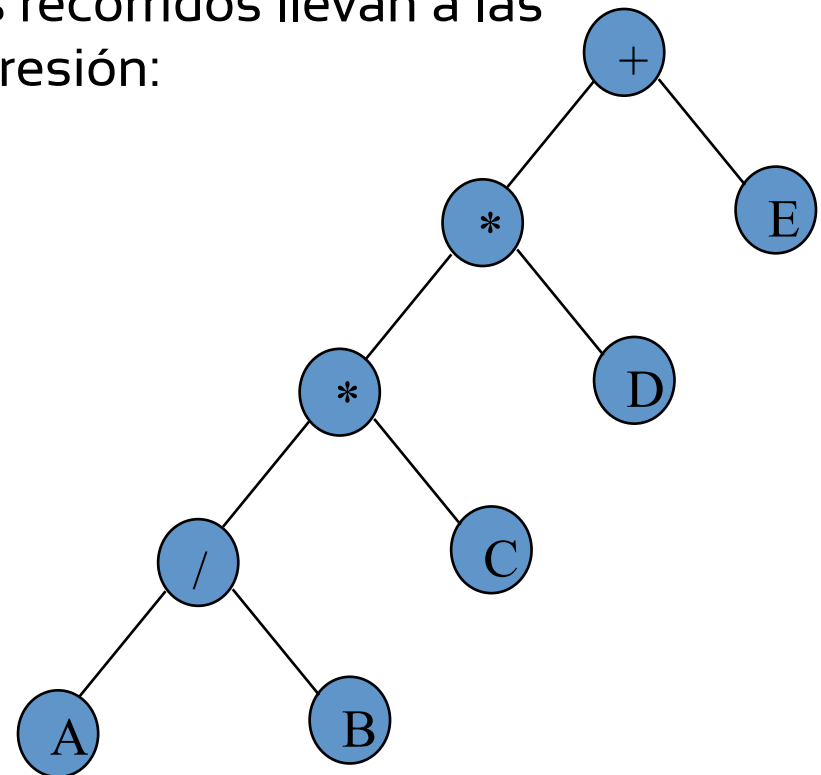
¿De donde vienen los nombres de los recorridos?

- Los recorridos se corresponden con las formas infija, postfija y prefija de escribir una expresión aritmética en un árbol binario.
- Dado el árbol binario, los diferentes recorridos llevan a las diferentes formas de escribir la expresión:

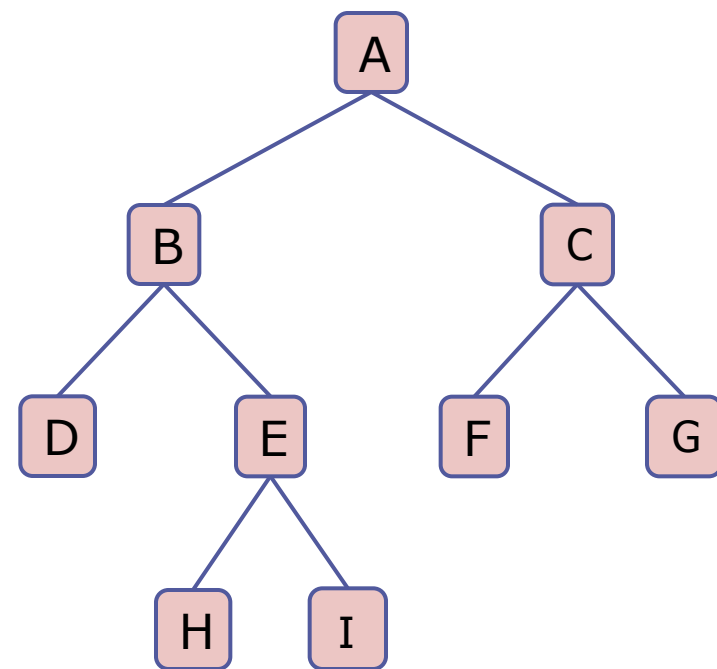
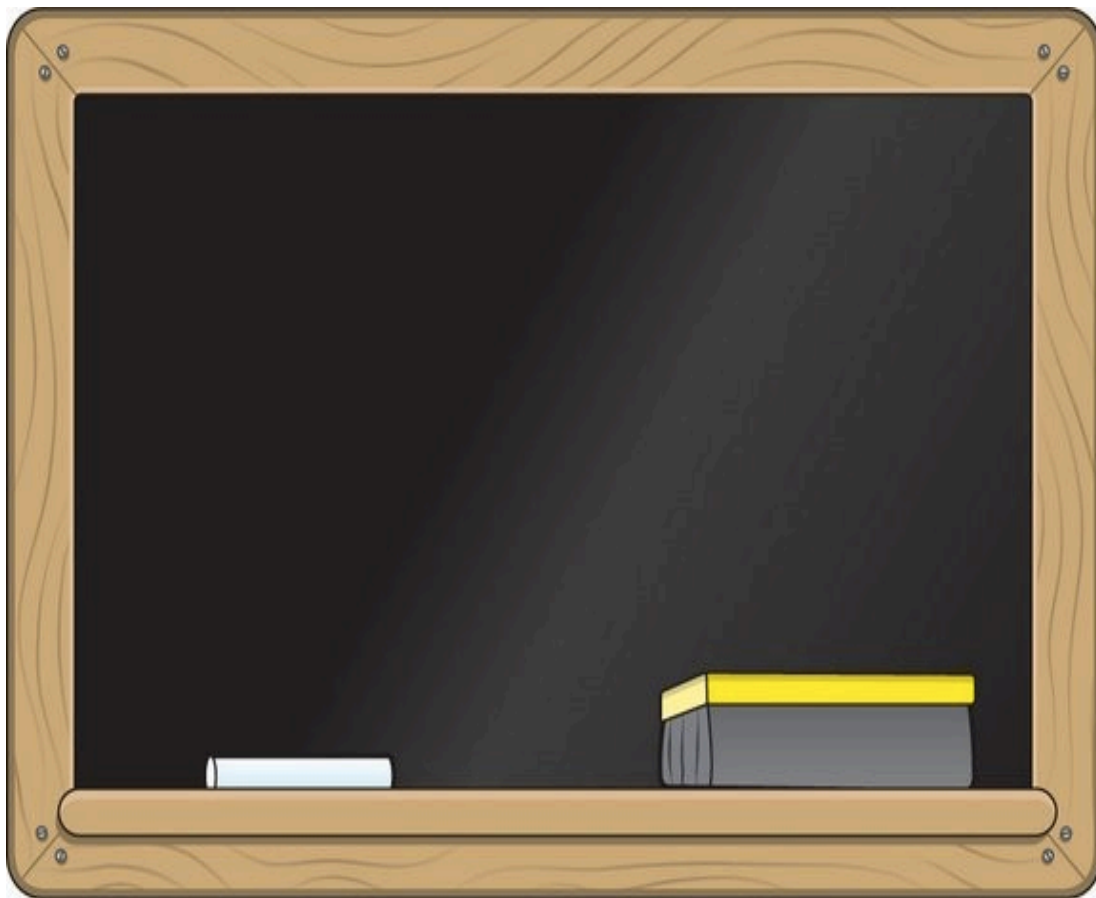
– **Preorden:** +**/ABCDE

– **Inorden:** A/B*C*D+E

– **Postorden:** AB/C*D*E+



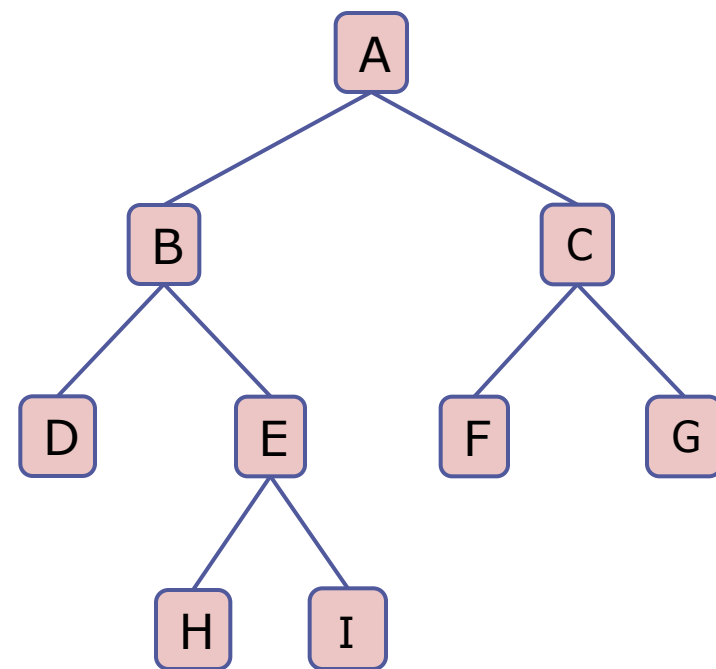
Recorrido Pre-Orden



**Recorrido
Pre-Orden:**
A,B,D,E,H,I,C,F,G

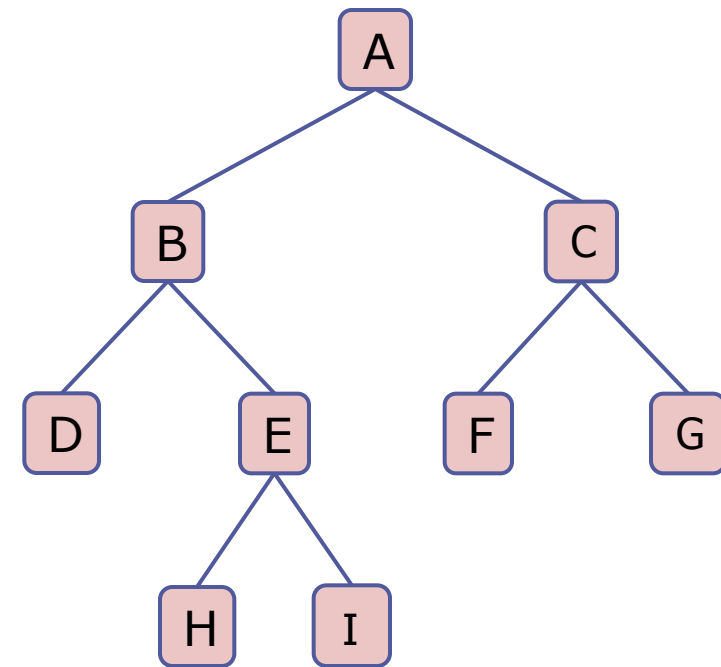
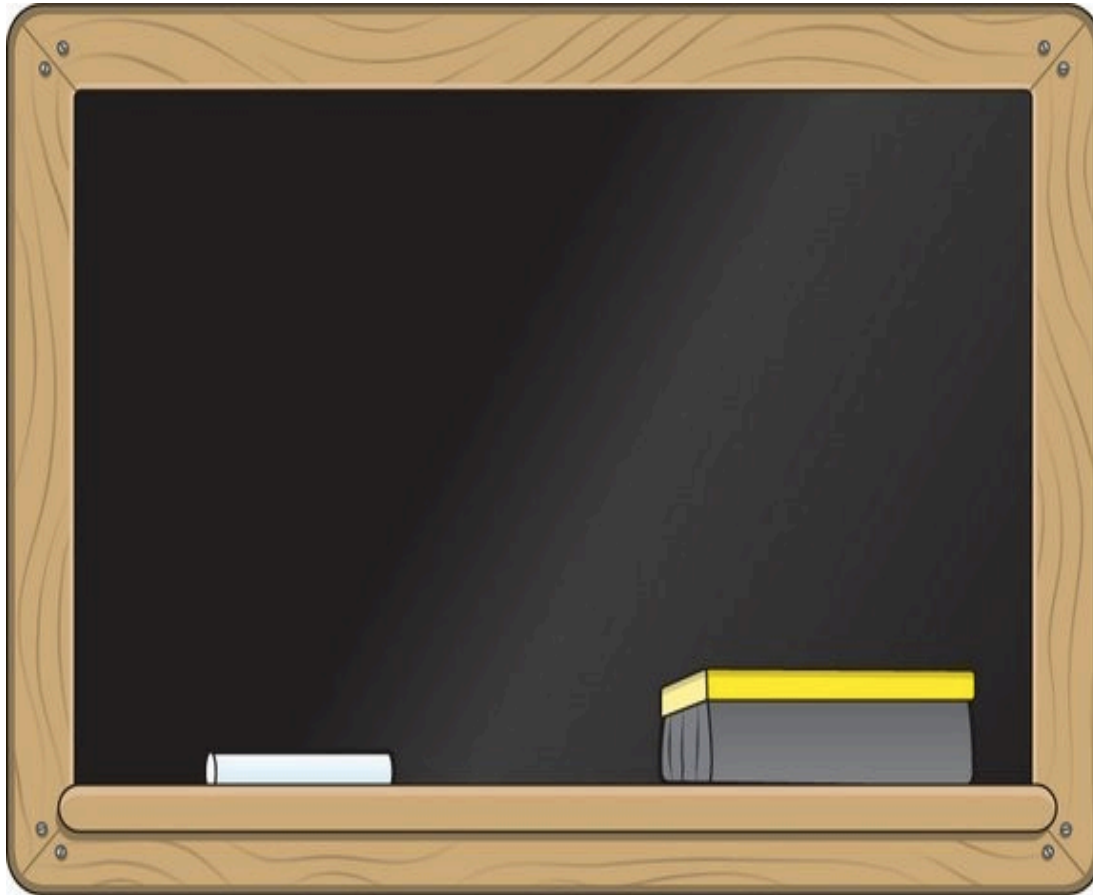
Recorrido Pre-Orden

```
function preorder(node):  
    //Input: root node of tree  
    //Output: None  
    visit(node)  
    if node has left child  
        preorder(node.left)  
    if node has right child  
        preorder(node.right)
```



**Recorrido
Pre-Orden:**
A,B,D,E,H,I,C,F,G

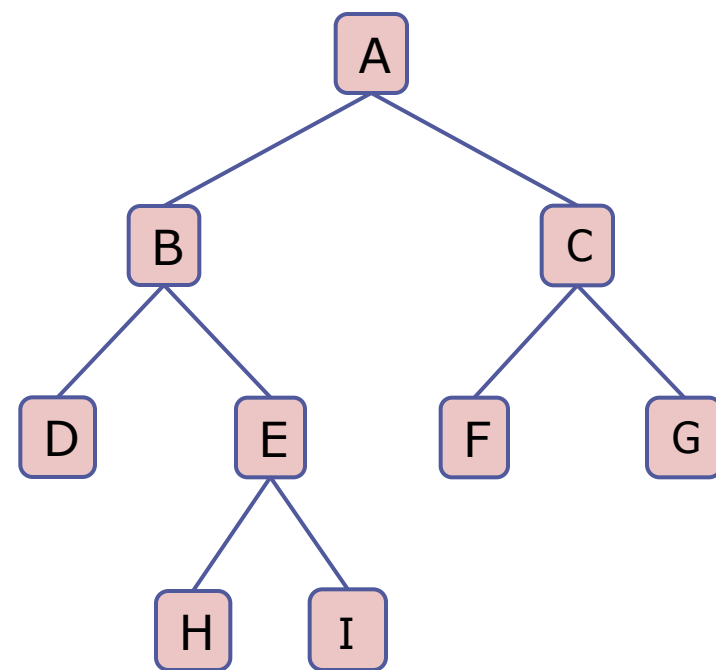
Recorrido Post-Orden



**Recorrido
Post-Orden:**
D,H,I,E,B,F,G,C,A

Recorrido Post-Orden

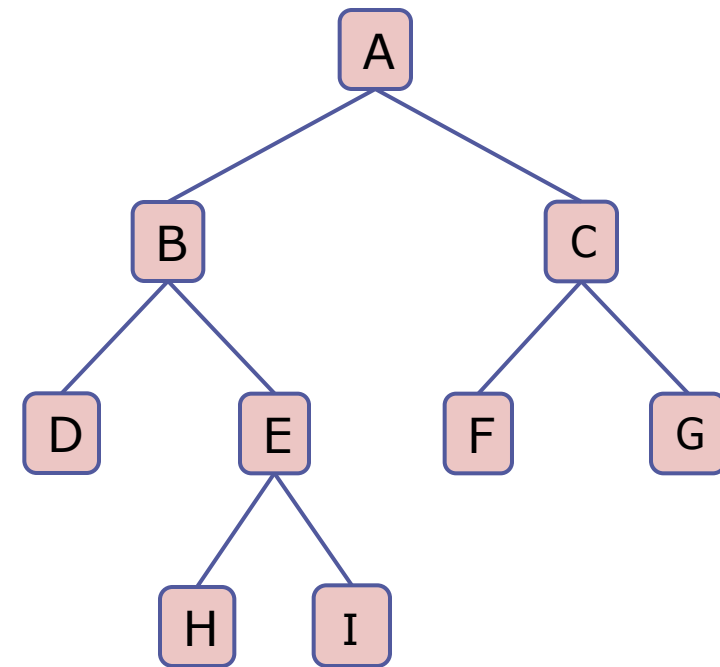
```
function postorder(node):  
    //Input: root node of tree  
    //Output: None  
    if node has left child  
        postorder(node.left)  
    if node has right child  
        postorder(node.right)  
    visit(node)
```



**Recorrido
Post-Orden:**
D,H,I,E,B,F,G,C,A

Recorrido In-Orden

```
function inorder(node):  
    //Input: root node of tree  
    //Output: None  
    if node has left child  
        inorder(node.left)  
    visit(node)  
    if node has right child  
        inorder(node.right)
```



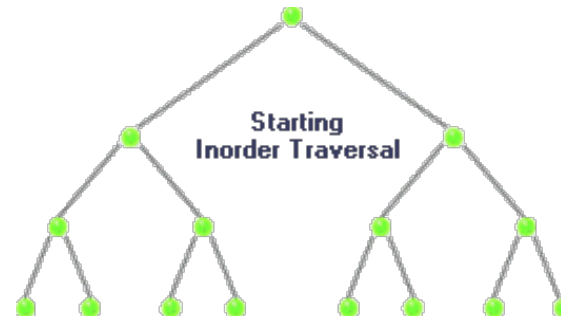
**Recorrido
In-Orden :**
D,B,H,E,I,A,F,C,G

Visualización de los recorridos

Pre-Orden



In-Orden



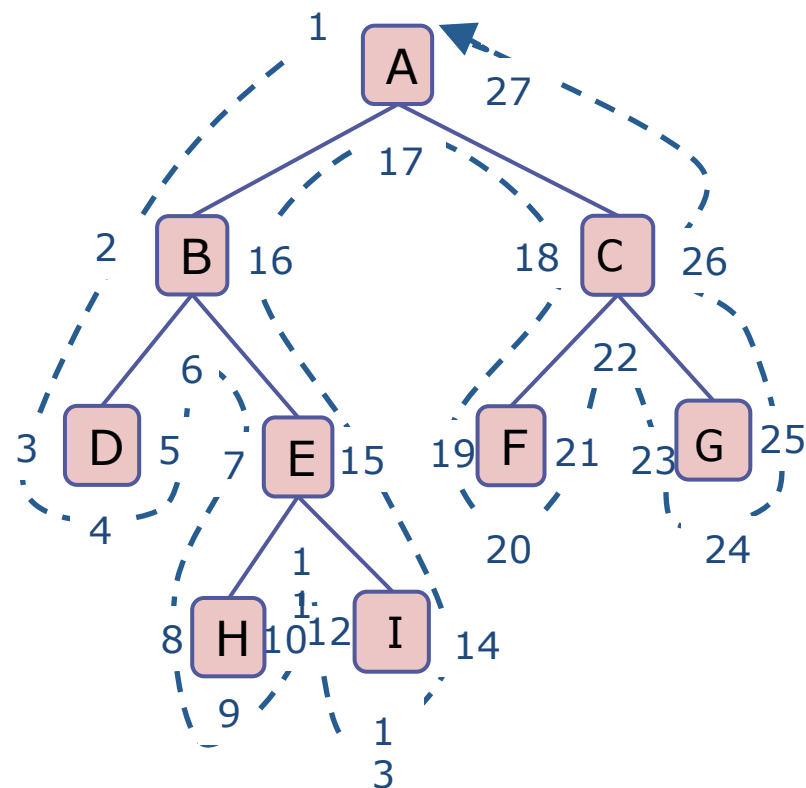
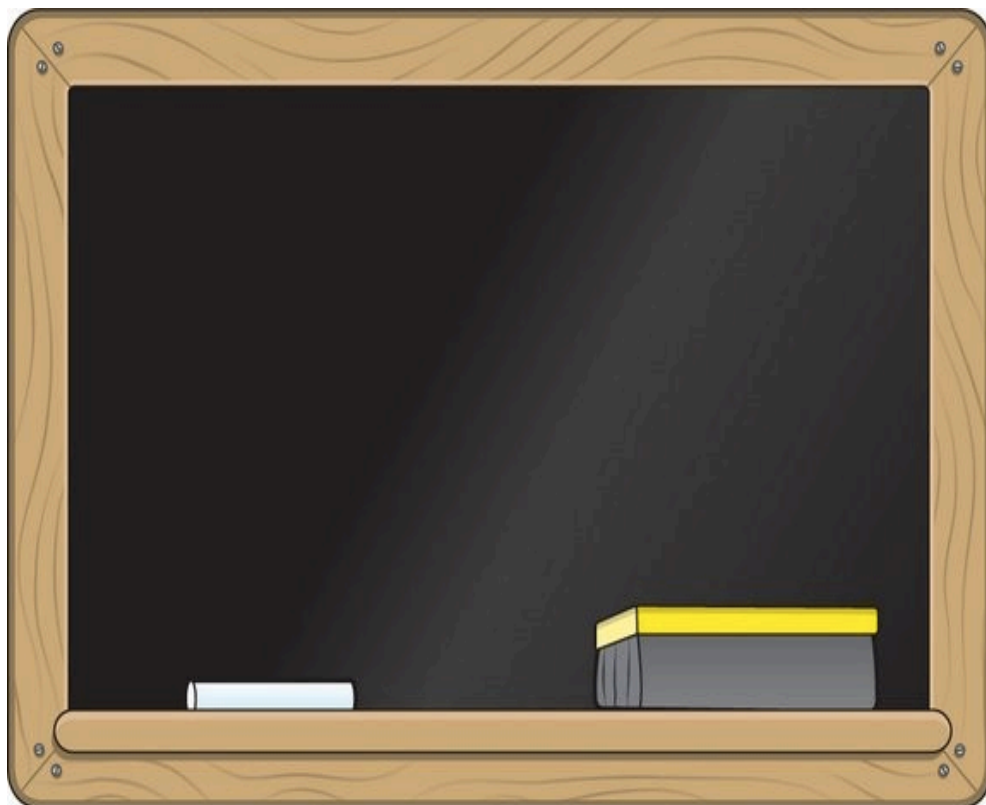
Post-Orden



Recorrido de Euler

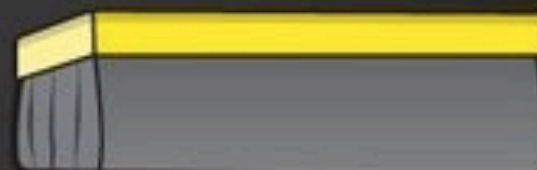
- Es el recorrido genérico de un árbol binario.
- Incluye los casos especiales de recorridos **preorden**, **inorden** y **postorden**.
- Cada nodo es visitado 3 veces!
 - Por la izquierda (**preorden**)
 - Por debajo (**inorden**)
 - Por la derecha (**postorden**)

Recorrido de Euler: Pseudo-código

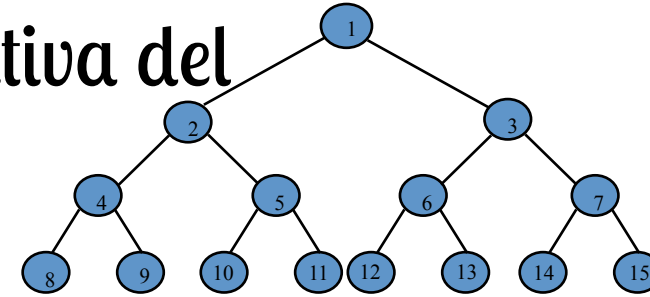


Recorrido de Euler: Pseudo-código

```
function eulerTour(node):  
    // Input: root node of tree  
    // Output: None  
  
    visitLeft(node) // Pre-order  
  
    if node has left child:  
        eulerTour(node.left)  
  
    visitBelow(node) // In-order  
  
    if node has right child:  
        eulerTour(node.right)  
  
    visitRight(node) // Post-order
```



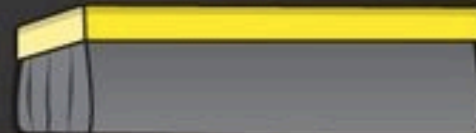
¿Cómo sería la implementación iterativa del recorrido inorden (LVR)?



Pasos:

- i) Guardar el camino (1,2,4,8)
- ii) Visitar el nodo (sacando el último por donde hemos pasado - 8), camino (1,2,4)
- iii) Moverse un nodo a la derecha (no hay)
- iv) Sacar el siguiente (4) del camino que quedará: (1,2)
- v) Visitar 4
- vi) Moverse a la derecha (9). Camino(1,2,9)
- vii) Descender - no podemos.
- viii) Visitar el último (9). Camino: (1,2)
- ix) Moverse a la derecha (no podemos).
- x) ...

¿Qué estructura necesitamos para guardar el camino?



¿Cómo sería la implementación iterativa del recorrido inorden (LVR)?

```
def IterInorden(self):  
    s = Stack()  
    current = self._getRoot()  
    while(s.isEmpty()!=false):  
        if(current!= None):  
            s.push(current)  
            current = current.getLeft()  
        else:  
            current = s.top();  
            s.pop();  
            visit(current);  
            current = current.getRight()
```

Quando utilizar los distintos tipos de recorridos?

- ¿Como sabemos cual es el mejor método para recorrer el árbol?
- Algunas veces no importa, pero normalmente hay un método que nos permite solucionar el problema de forma mucho más sencilla y eficiente.

Problema 1

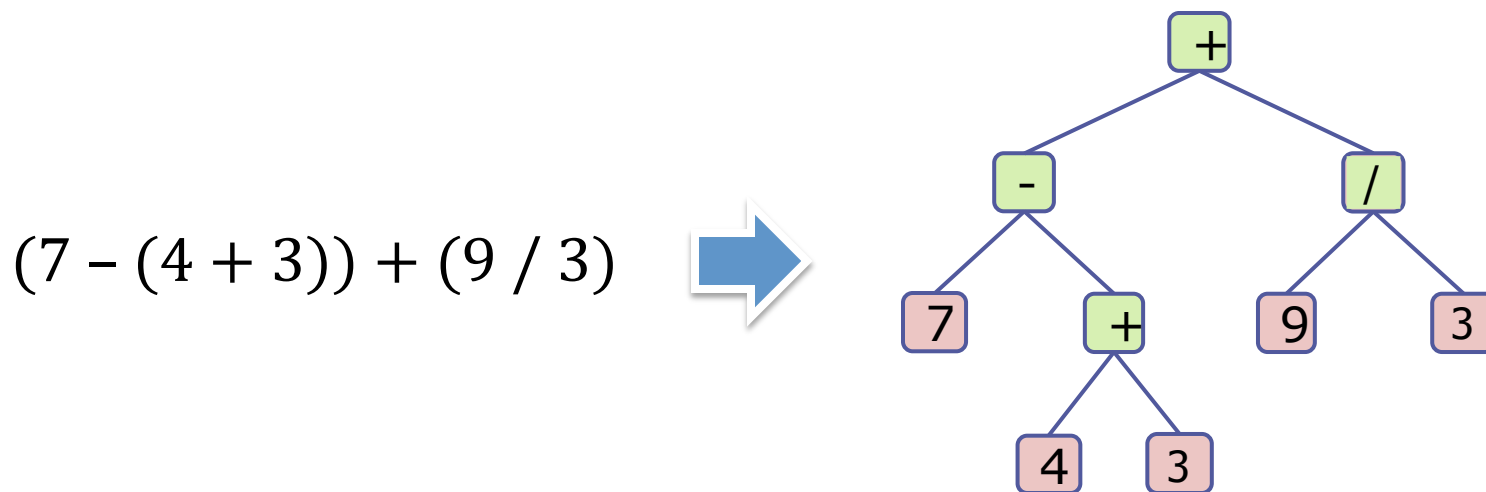
- Indicar en cada nodo cuantos descendientes tiene.
- Mejor recorrido: **post-orden**
 - Es fácil calcular el número de descendientes de un nodo si ya sabemos cuantos descendientes tienes sus hijos.
 - El post-orden visita los nodos hijos antes que nodo padre, esto es exactamente lo que queremos.

Problema 2

- Dado la raíz de un árbol, determinar si un árbol es perfecto.
- Mejor recorrido: **breadth-first**
 - La mejor solución se encuentra explorado el árbol nivel a nivel.
 - Podemos hacer seguimiento del nivel actual y contar el número de nodos que tiene
 - Cada nivel tiene que tener el doble de nodos que el nivel anterior.

Problema 3

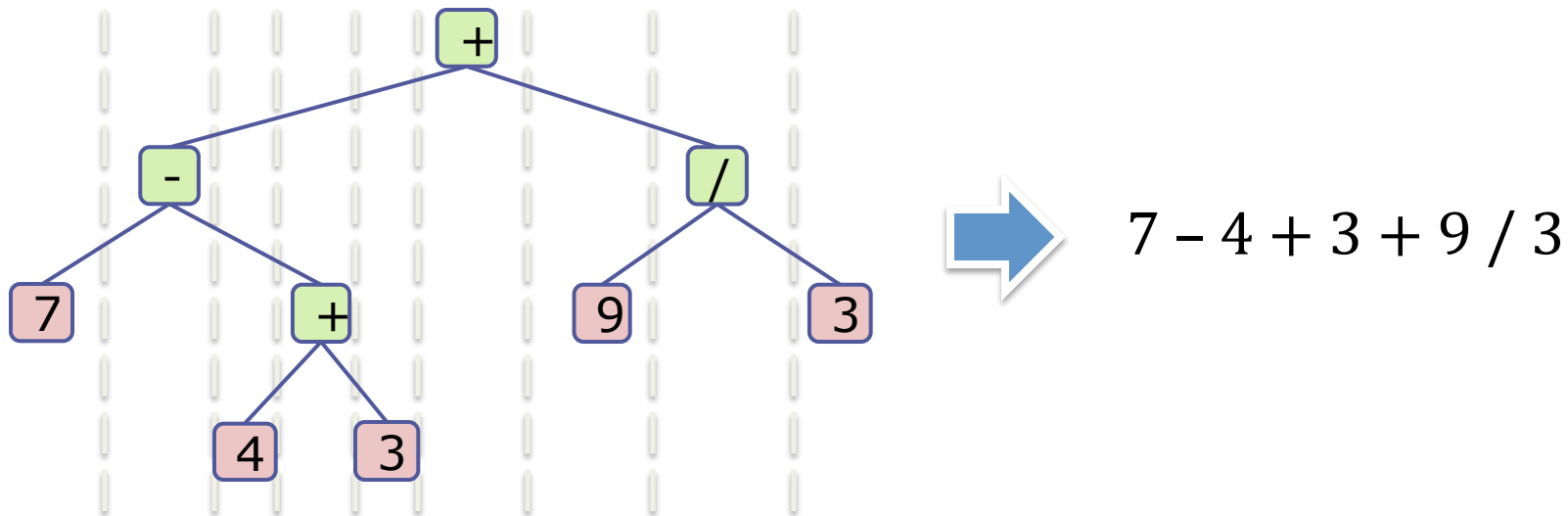
- Dado la árbol que representa una expresión aritmética, evaluar su resultado:



- Mejor recorrido: **post-orden**
 - Para evaluar la expresión aritmética, primero debemos evaluar las sub-expresiones de cada hijo

Problema 4

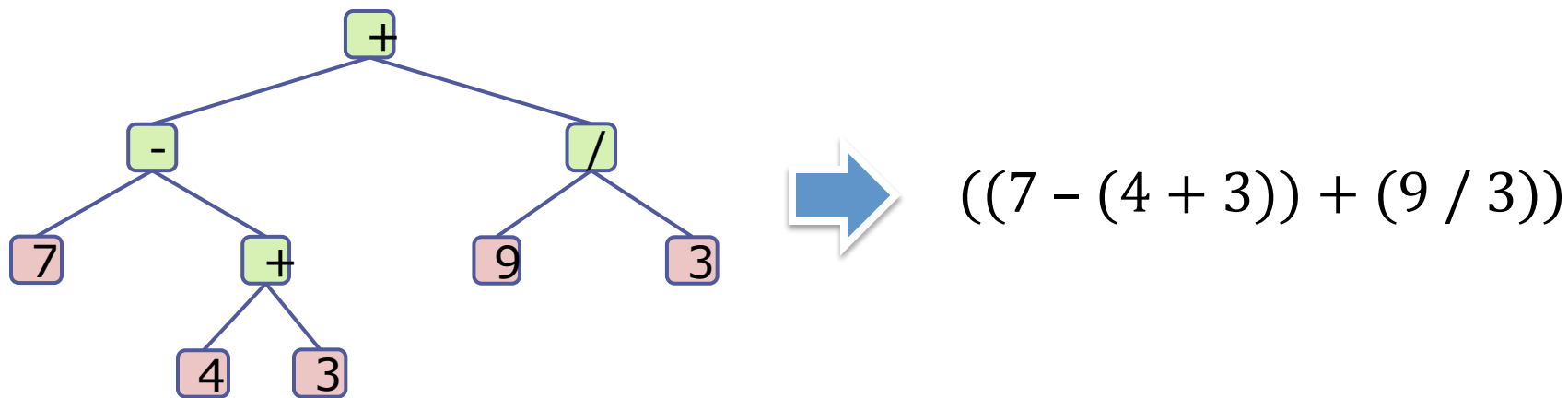
- Dado una expresión aritmética, imprimir su resultado sin paréntesis



- Mejor recorrido: **in-orden**
 - El recorrido in-orden nos da los nodos de izquierda a derecha.

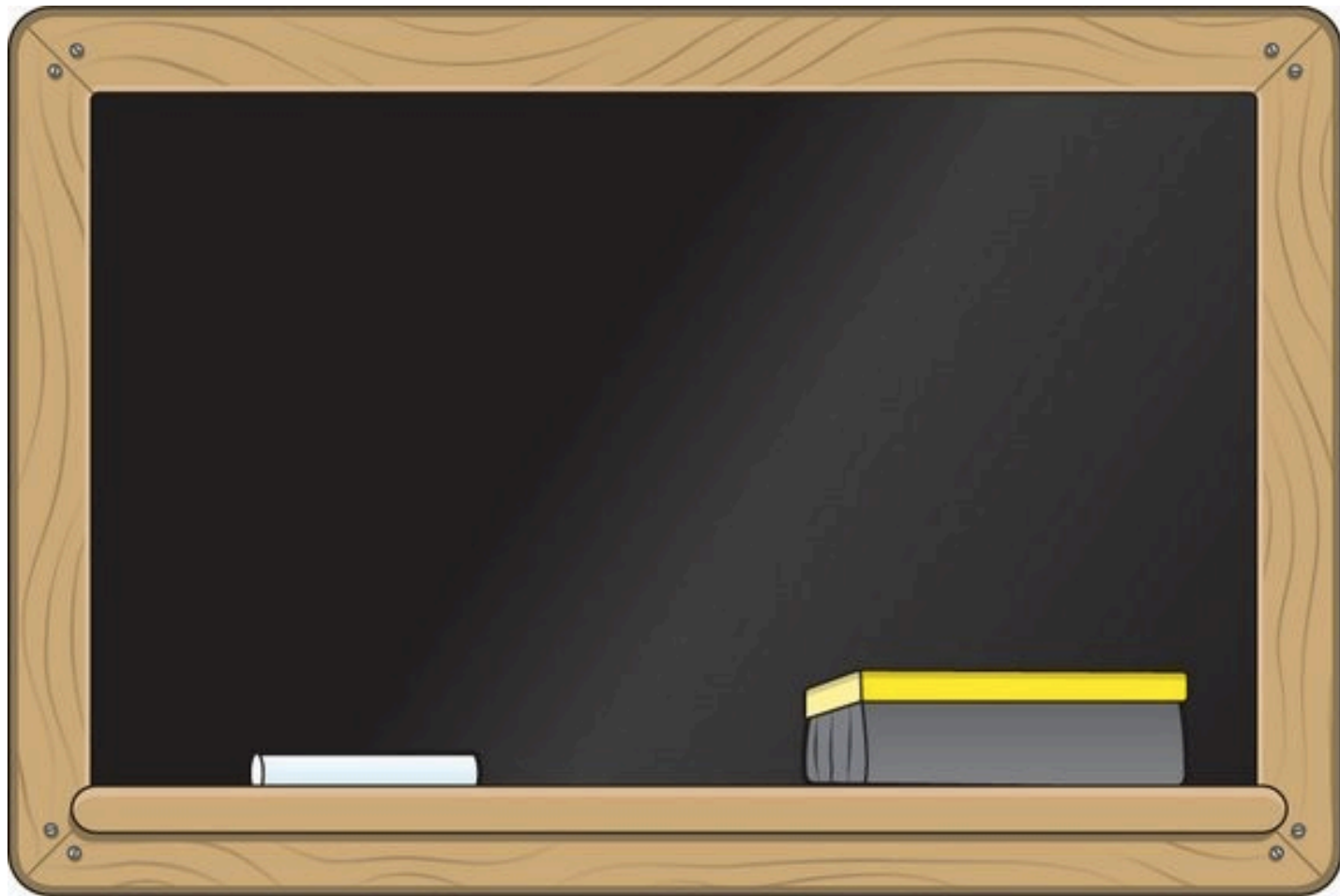
Problema 5

- Dado un árbol que representa una expresión aritmética, imprimir la expresión con paréntesis.



- Mejor Recorrido: **Recorrido de euler**
 - Si el nodo es un nodo interno (un operador):
 - Para el pre-orden imprimir "("
 - Para el in-orden imprimir el operador
 - Para el post-orden imprimir ")"
 - Si el nodo es una hoja (un número)
 - No hacer nada para el pre-orden ni post-orden
 - Para el in-orden, imprimir el número

¿Cómo contar el número de nodos de un árbol?



¿Con que estructura es mas rápido?

¿Cómo contar el número de nodos de un árbol?

- Análisis de la solución recursiva:
 - ¿Cuál es el árbol más pequeño sobre el que se puede contar?
 - Respuesta: un árbol vacío, en cuyo caso la cantidad de los elementos es cero.
 - ¿Cómo obtener la cantidad de elementos que tiene un árbol si se sabe cuántos elementos tiene el subárbol izquierdo y el subárbol derecho?
 - Respuesta: $1 + \text{cantidad de elementos del subárbol izquierdo} + \text{cantidad de elementos del subárbol derecho}$

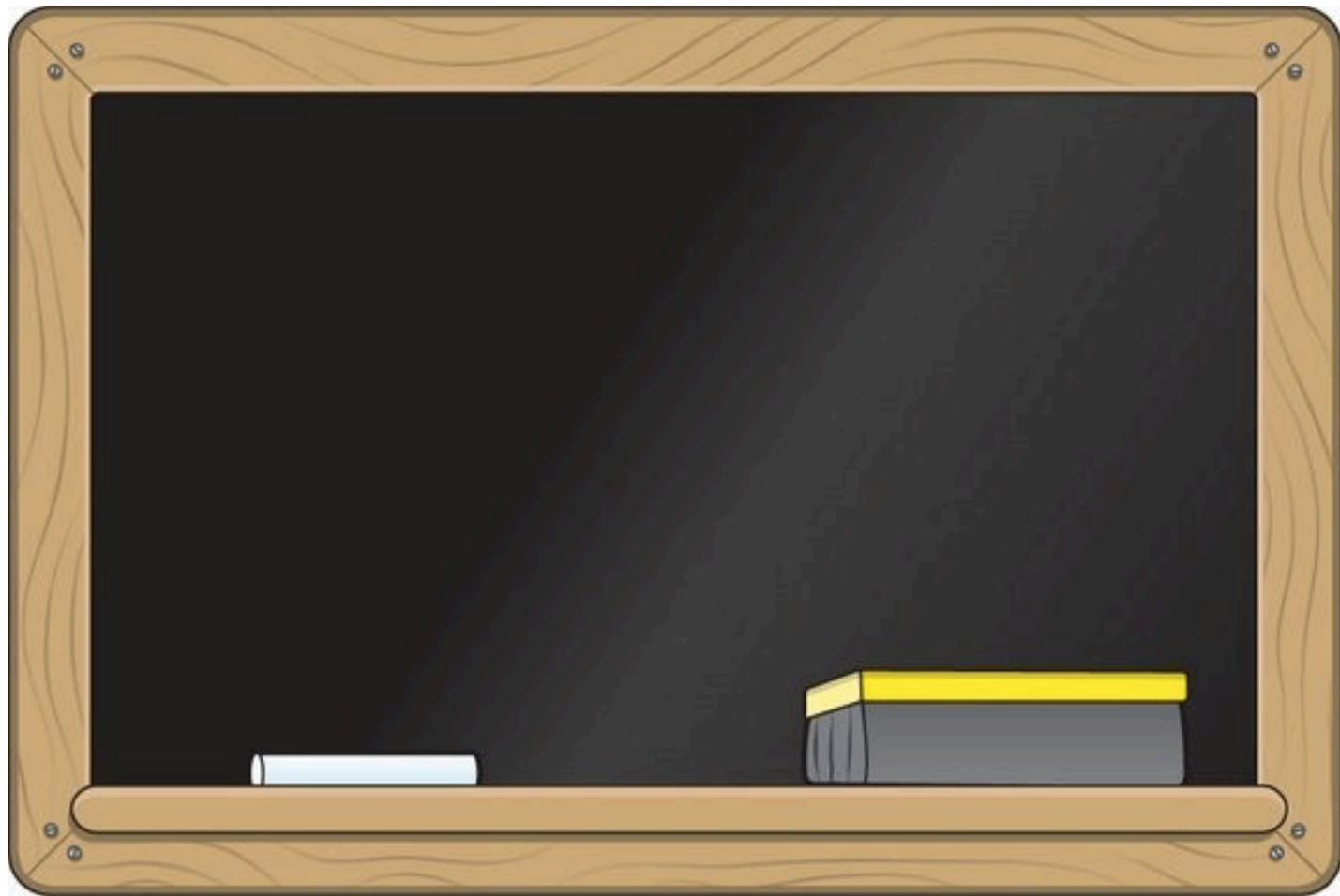
¿Cómo contar el número de nodos de un árbol?

Implementación:

```
def contarNodos(self):  
    return self._contarNodos(self._root)  
  
def _contarNodos(self, node):  
    if (node==None):  
        return 0  
    left  = self._contarNodos(node.getLeft())  
    right = self._contarNodos(node.getRight())  
    return 1+left+right
```

#¿Cuál es similitud y diferencia con len() de la implmenatación con vectores?

Comparar dos árboles



Copiar recursivamente un árbol

