



Tema 4 Estructures No Lineals: Arbres

Sessió Teo 9

Maria Salamó Llorente
Estructura de Dades

Grau en Enginyeria Informàtica
Facultat de Matemàtiques i Informàtica,
Universitat de Barcelona



Contingut

Sessió Teoria 7 (Teo 7)

4.1 Introducció als arbres

4.2 Arbres binaris

Sessió Teoria 8 (Teo 8)

4.3 Arbres binaris de cerca

Sessió Teoria 9 (Teo 9)

4.4. Recorreguts en arbres binaris

Sessió Teoria 10 (Teo 10)

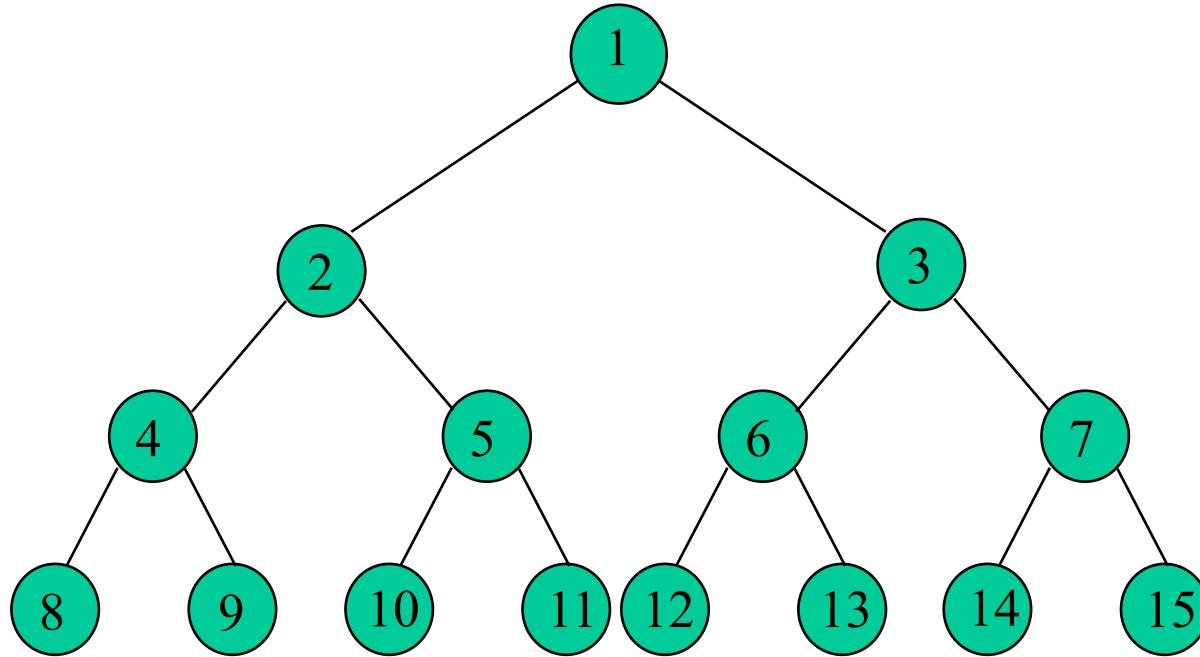
4.5. Arbres AVL



4.4 Recorreguts en arbres binaris

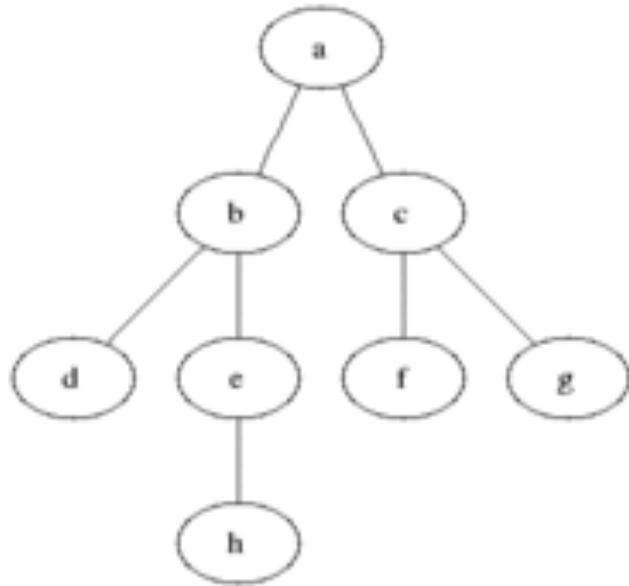
Com recórrer un arbre binari?

- Quines estratègies es poden seguir?
 - Top to Bottom? Left to Right?

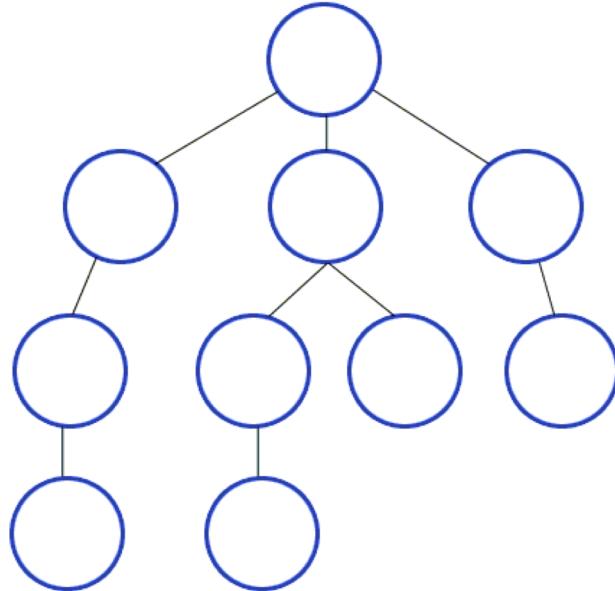


Recorregut Amplada vs. Profunditat

Breadth-First BFT



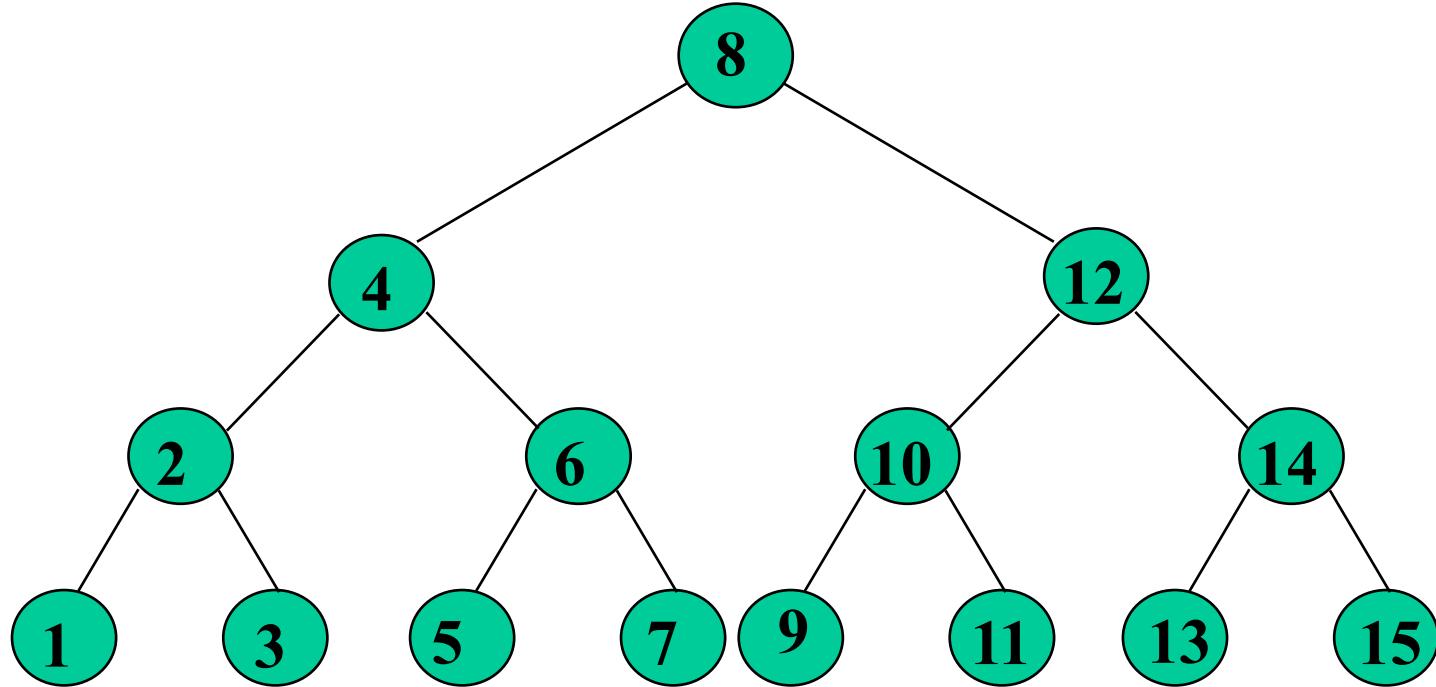
Depth-First DFT



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

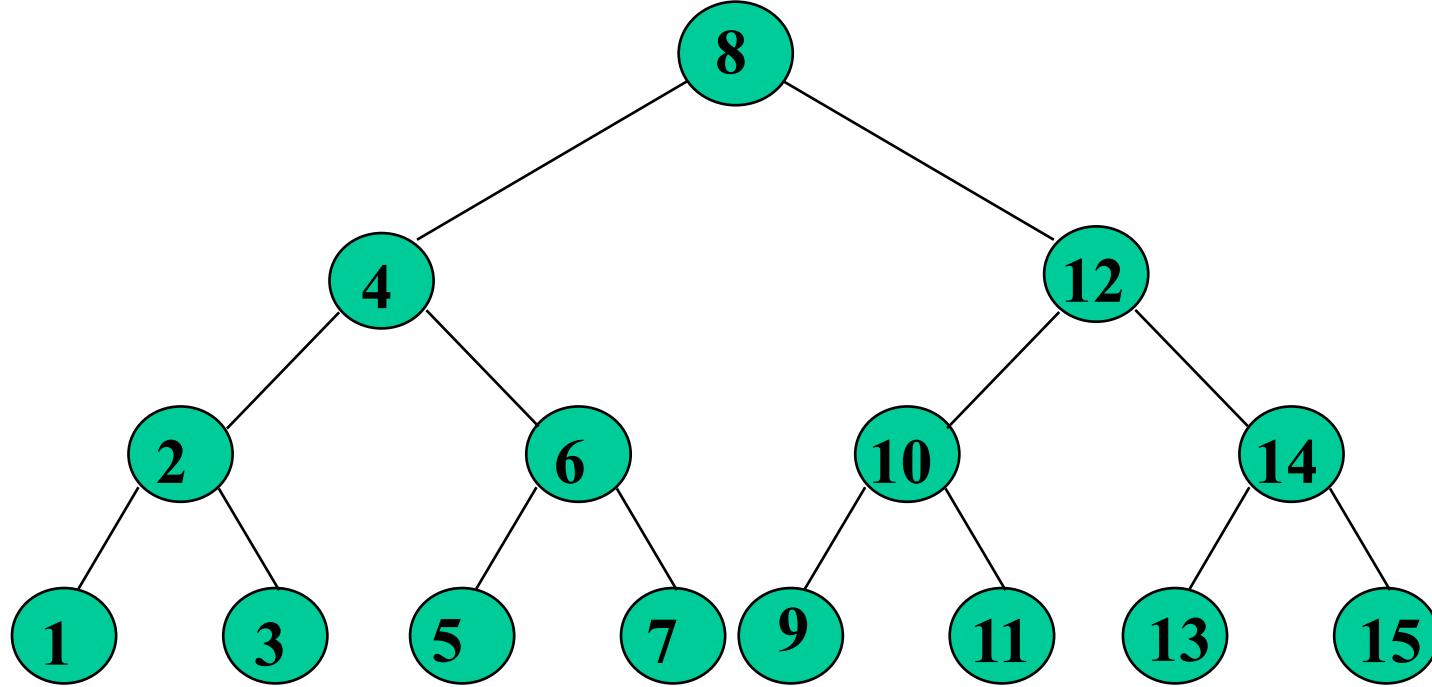
DFT gif: <https://upload.wikimedia.org/wikipedia/commons/7/7f/Depth-First-Search.gif>

Com recórrer un arbre binari?



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

Com recórrer un arbre binari?

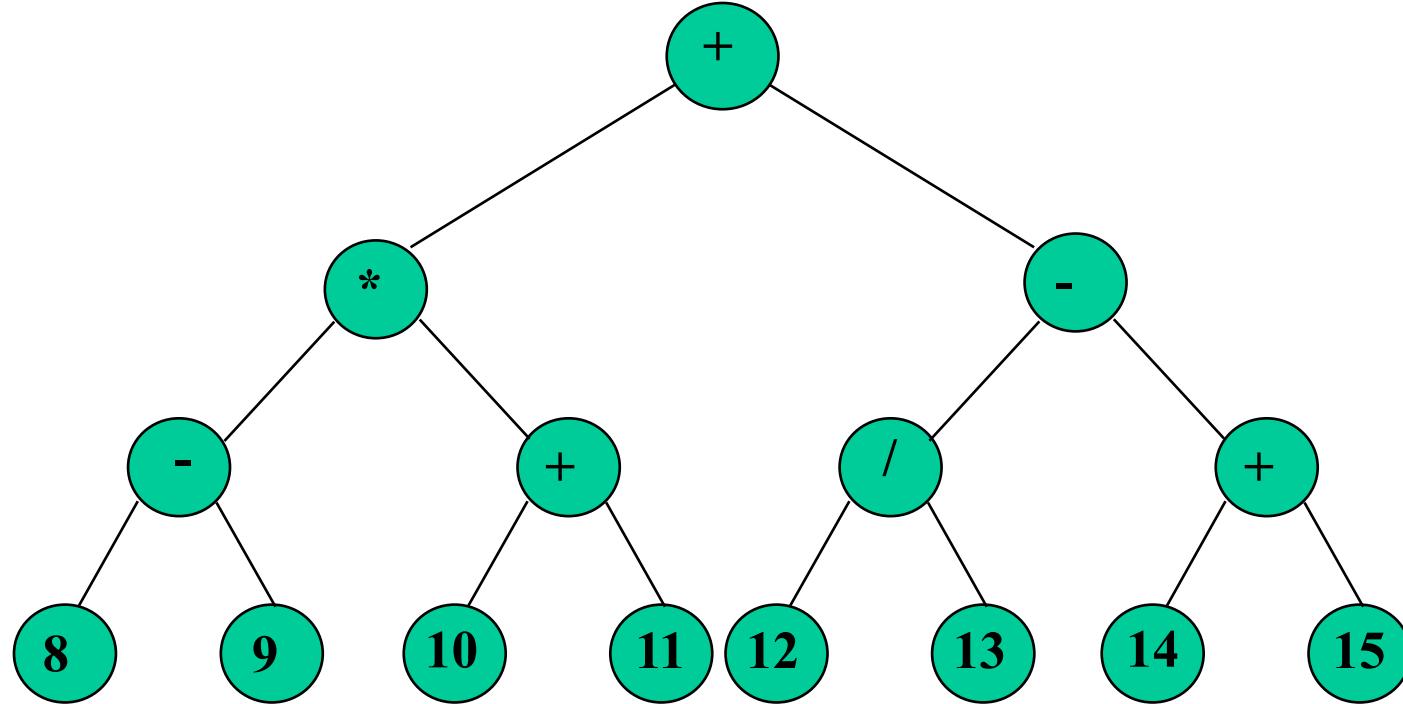


Recorregut en profunditat

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

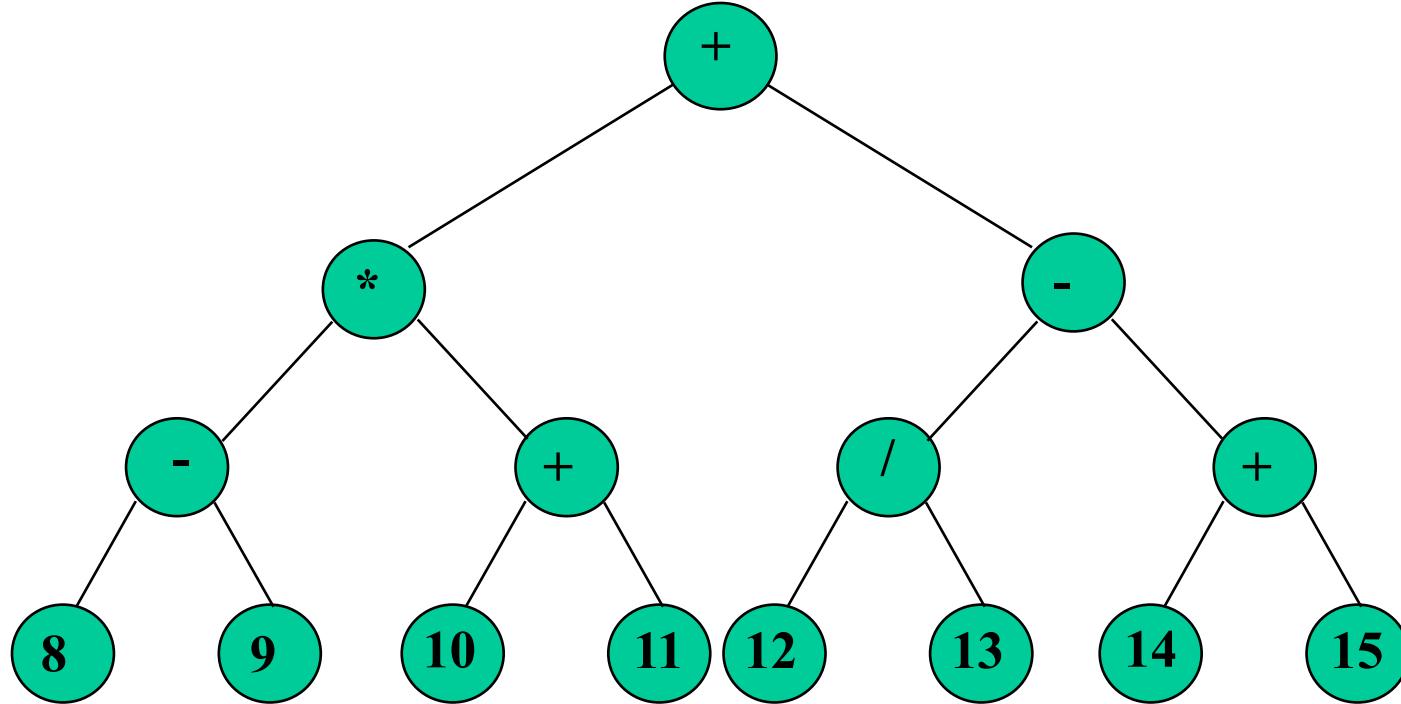
PreOrdre 8 - 4 - 2 - 1 - 3 - 6 - 5 - 7 - 12 - 10 - 9 - 11 - 14 - 13 - 15

Com recórrer un arbre binari?



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

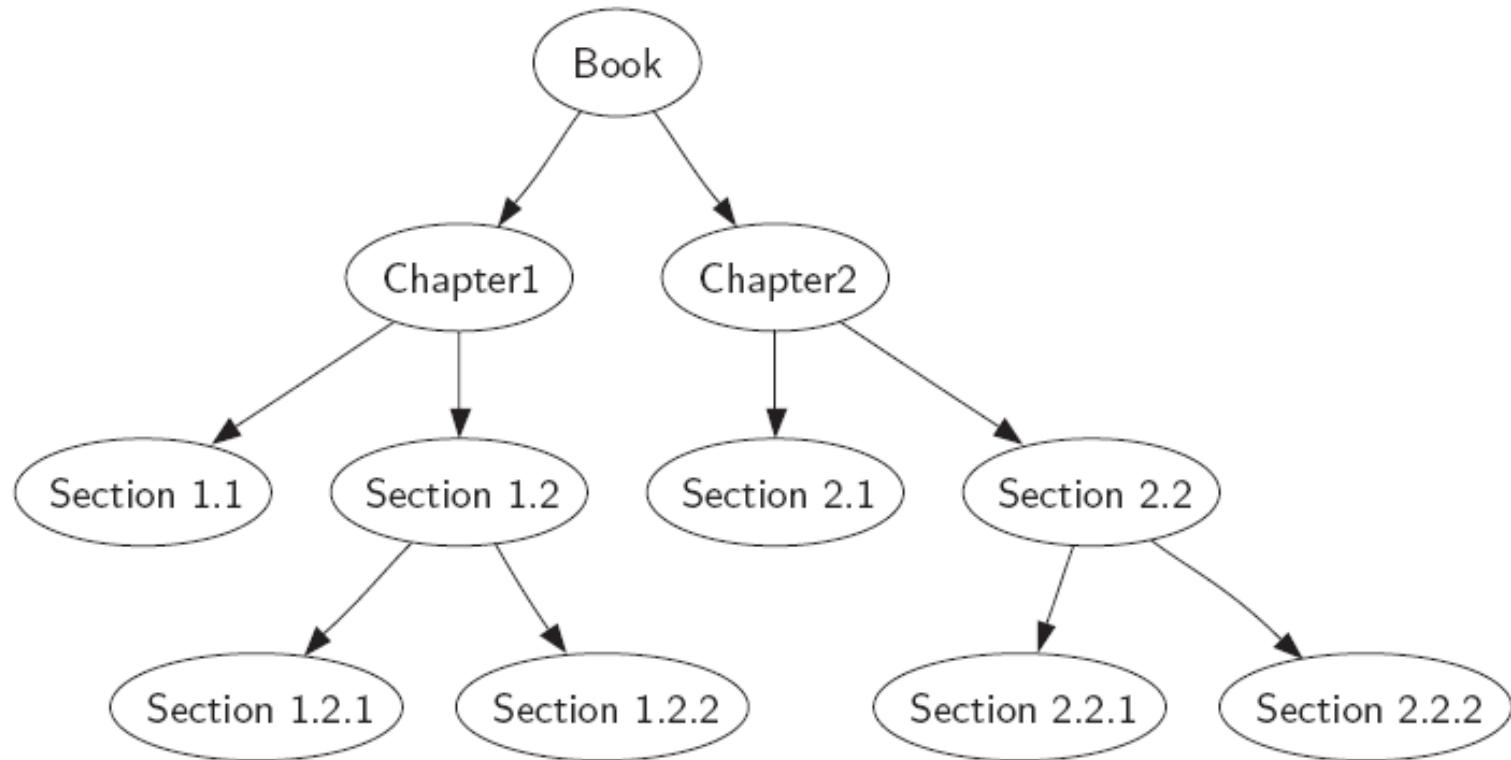
Com recórrer un arbre binari?



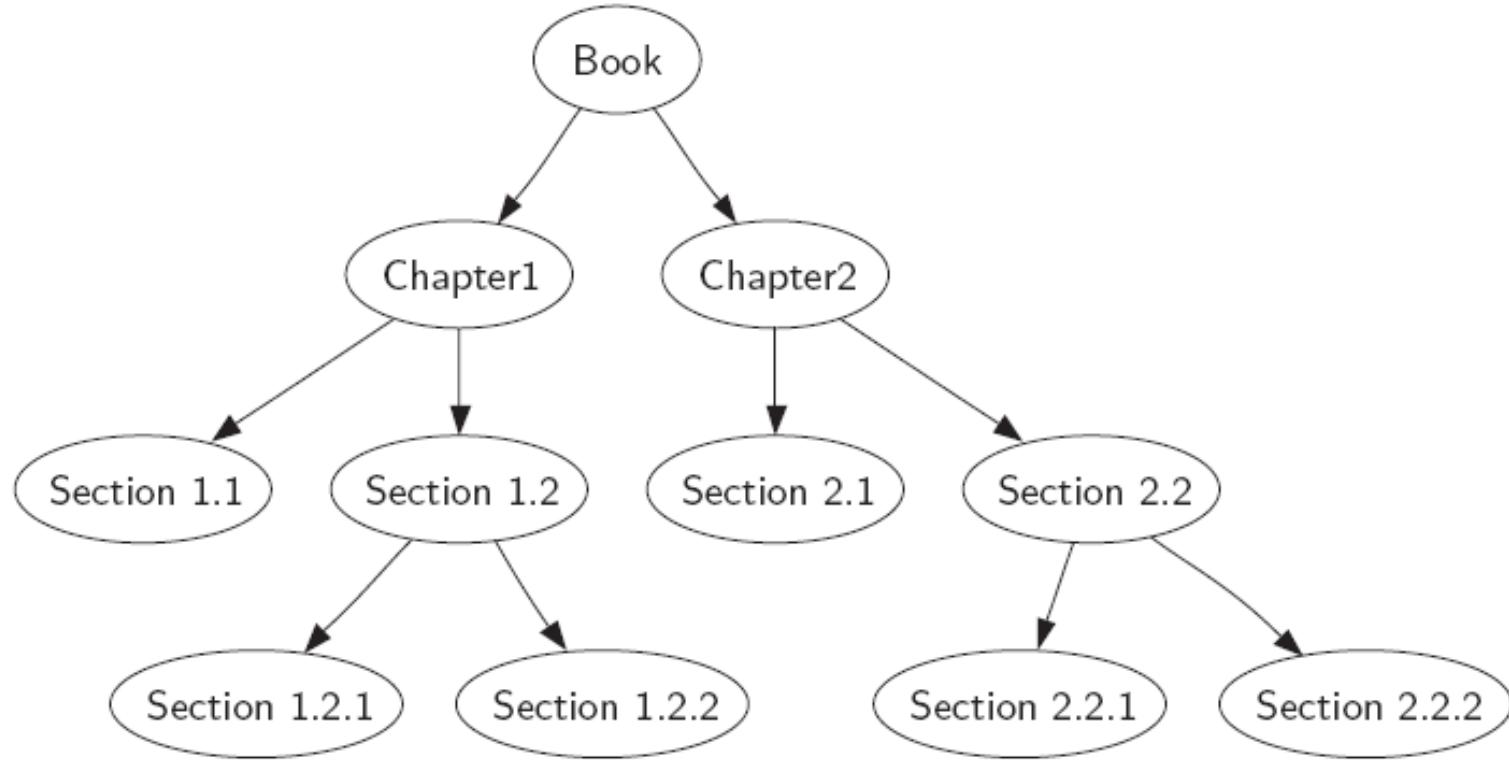
Recorregut en profunditat

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

Com recórrer un arbre binari?



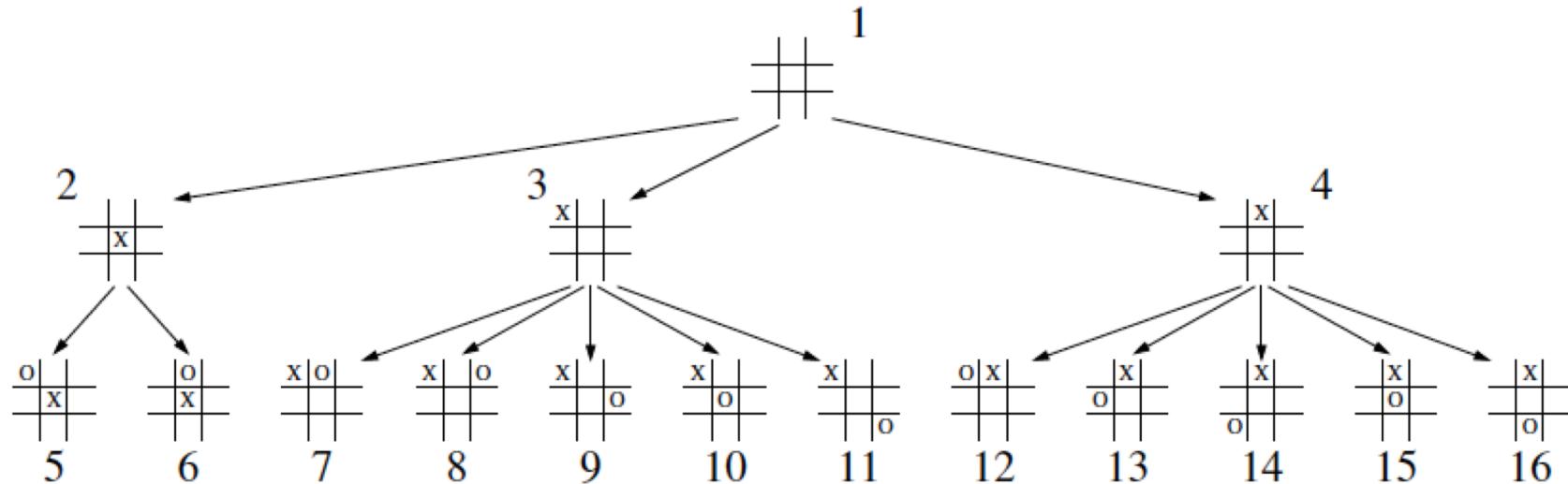
Com recórrer un arbre binari?



Recorregut en profunditat

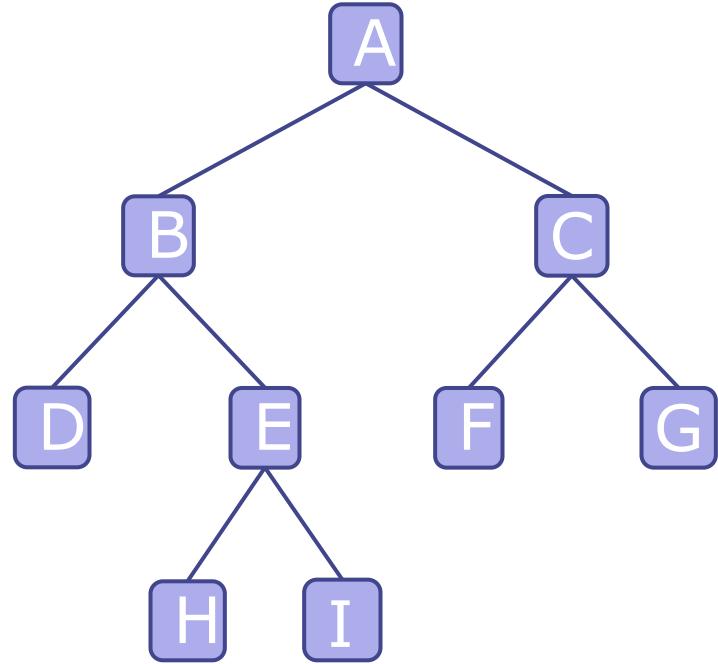
Recorregut en amplada

- Recorregut **breadth-first** (amplada)
 - El recorregut en amplada és molt comú en el desenvolupament de jocs. Un arbre de joc representa els possibles moviments en el joc que pot fer un jugador o un ordinador, essent l'arrel de l'arbre el moviment inicial.



Recorregut en amplada: Breadth-first

- Es comença des del node arrel, visitem tots els seus fills, després visitem tots els seus néts, després els besnéts, després



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



Recorregut en amplada: Breadth-first

Estratègia general: Utilitzar una **cua** (Queue) per fer el seguiment de l'ordre dels nodes

Pseudo-codi:

```
Algorithm 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 cua garanteix que tots els nodes d'un mateix nivell seran visitats abans que qualsevol dels seus fills

Recorregut en amplada: Breadth-first

Algorithm bft(root) :

```
Q = new Queue()  
enqueue root  
while Q is not empty:  
    node = Q.dequeue()  
    visit(node)  
    enqueue node's left & right children
```

Inici: Q = { }, Pas1: Q = {A}

Bucle, it 1: Visita A, Q= {B, C}

Bucle, it 2: Visita B, Q= {C, D, E}

Bucle, it 3: Visita C, Q= {D, E, F, G}

Bucle, it 4: Visita D, Q={E, F, G}

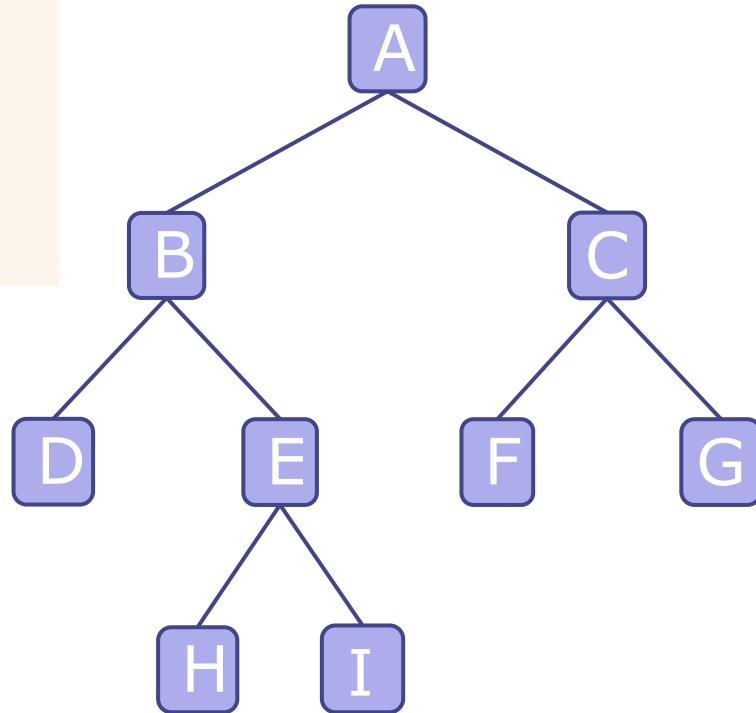
Bucle, it 5: Visita E, Q={F, G, H, I}

Bucle, it 6: Visita F, Q={G, H, I}

Bucle, it 7: Visita G, Q={H, I}

Bucle, it 8: Visita H, Q={ I}

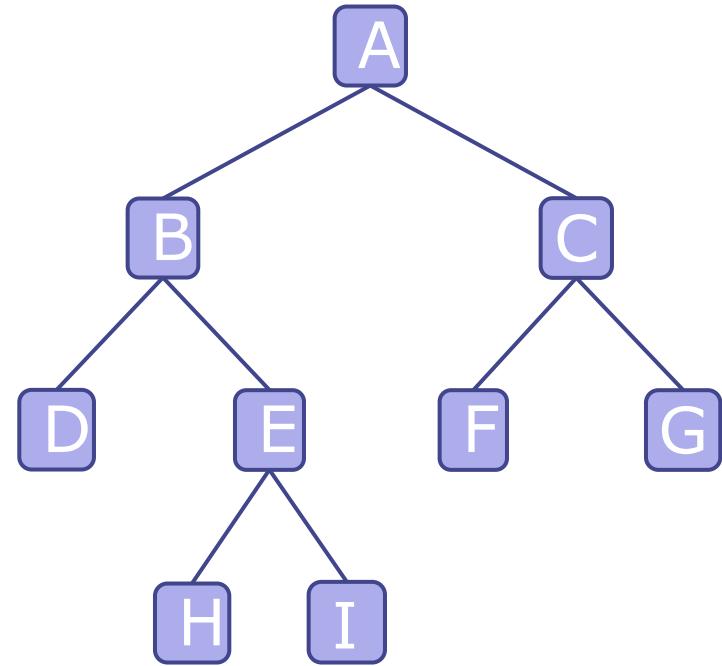
Bucle, it 9: Visita I, Q = { }



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

Recorregut en profunditat: depth-first

- Es comença des del node arrel, s'explora cada branca el més profundament possible abans de realitzar backtracking
- Això pot generar ordenacions molt diferents dependent de quina branca es visita primer
 - Normalment es decideix visitar d'esquerra a dreta



Recorregut en profunditat:

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

o

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



Recorregut en profunditat: Depth-first

- **Estratègia General:** Utilitzar una **pila** per fer un seguiment de l'ordre dels nodes.

Pseudo-codi:

```
Algorithm 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 garanteix que s'explora cada branca fins arribar a la fulla abans d'explorar una altra

Recorregut en profunditat: depth-first

Algorithm dft(root) :

```
S = new Stack()  
push root onto S  
while S is not empty:  
    node = S.pop()  
    visit(node)  
    push node's right and left children
```

S = {}, Pas 1: S = {A} // El top està al darrer element

Bucle, it 1: S = {}, Visita A, S = {C, B}

Bucle, it 2: S = {C}, Visita B, S = {C, E, D}

Bucle, it 3: S = {C, E}, Visita D, S = {C, E}

Bucle, it 4: S = {C}, Visita E, S = {C, I, H}

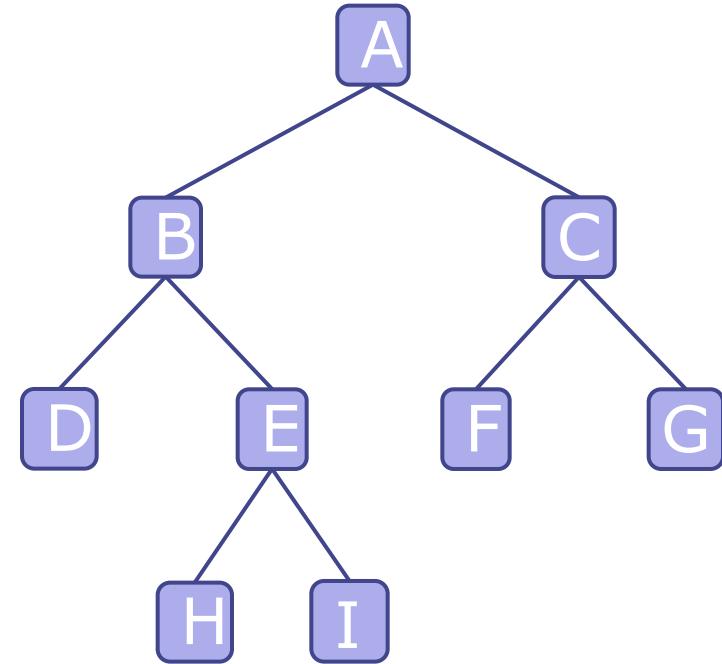
Bucle, it 5: S = {C, I}, Visita H, S = {C, I}

Bucle, it 6: S = {C}, Visita I, S = {C}

Bucle, it 7: S = {}, Visita C, S = {G, F}

Bucle, it 8: S = {G}, Visita F, S = {G}

Bucle, it 9: S = {}, Visita G, S = {}



Recorregut en profunditat:

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

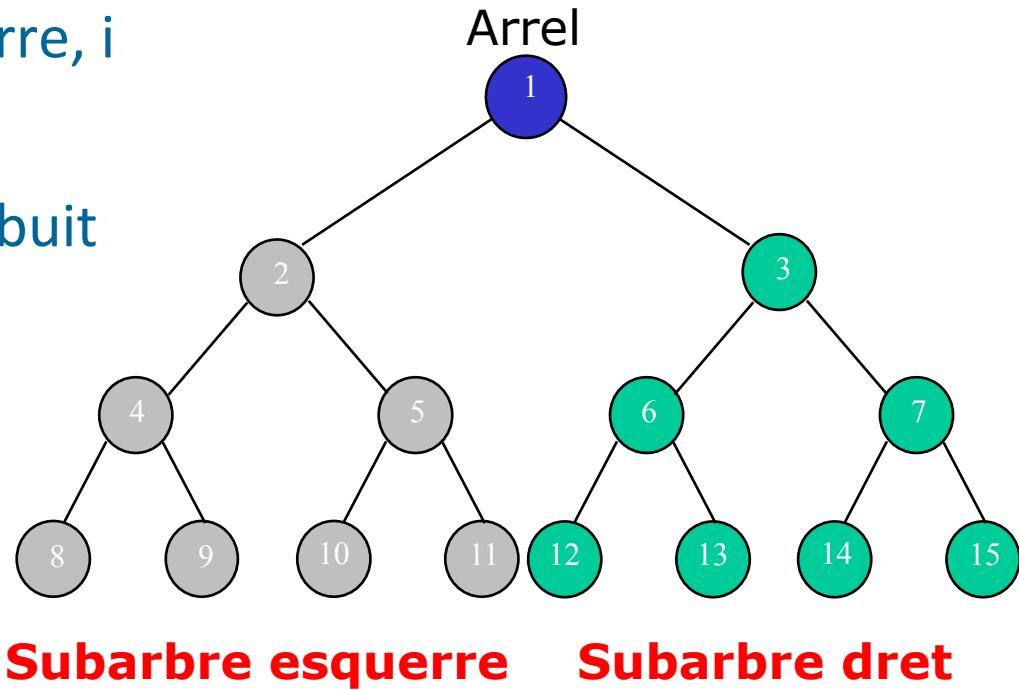


Altres solucions?

- Utilitzar una altra estructura de dades, ja sigui una pila o una cua significa que necessitarem més espai a memòria per tal de recorre l'arbre.
- Els mètodes de **recorregut en profunditat** són especialment òptims usant la **recursivitat**
 - Amb els mètodes recursius no ens caldrà definir una estructura de dades auxiliar per fer el recorregut

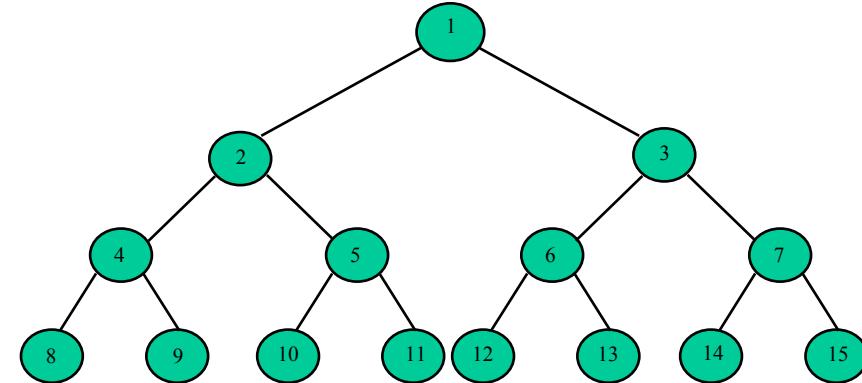
Recursivitat en arbres binaris

- Els mètodes de **recorregut en profunditat** són especialment òptims usant la **recursivitat**
- **Plantejament:**
 - **Cas general:** s'aplica l'operació recursiva sobre:
 - el subarbre esquerre, i
 - el subarbre dret
 - **Cas particular:** arbre buit



Com es pot recórrer un arbre binari en profunditat?

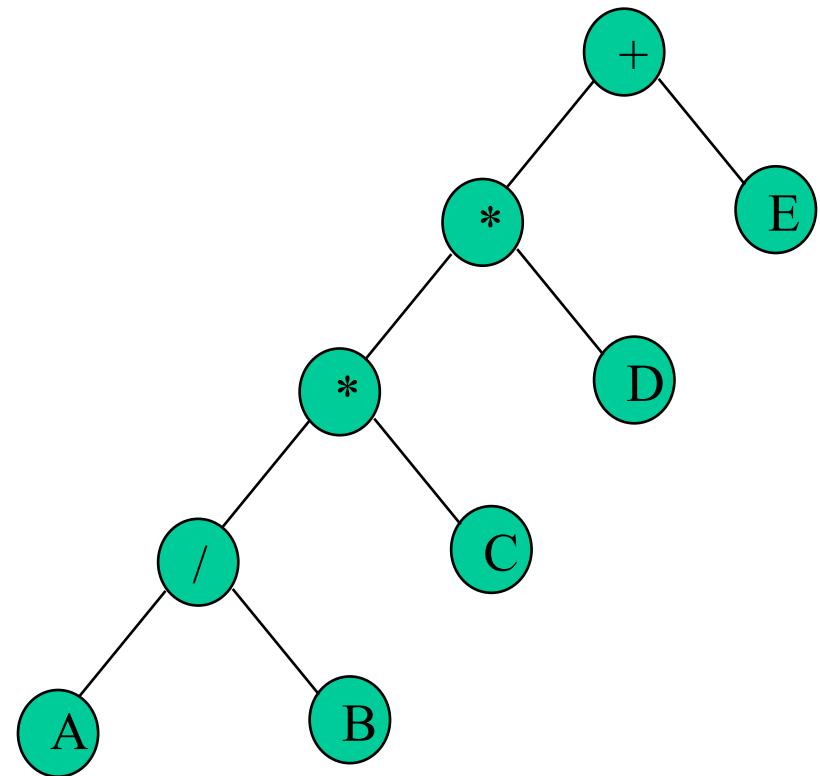
- Si denominem amb:
 - L: moviment a l'esquerra
 - V: “visitar” el node
 - R: moviment a la dreta
- Tenim sis combinacions possibles de recorregut:
 - **LVR, LRV, VLR, VRL, RVL, RLV**
- Si optem per realitzar primer el moviment a l'esquerra, tenim les tres primeres possibilitats
 - **LVR** l'anomenarem **inordre**,
 - **VLR** l'anomenarem **preordre**, i
 - **LRV** l'anomenarem **postordre**



Exemple

- Els diferents recorreguts sobre l'arbre d'expressions aritmètiques porten a les diferents maneres que hi ha d'escriure una expressió aritmètica (prefixa, infixa i posfixa)
- Donat el següent arbre binari:

- **Preordre:** +**/ABCDE
- **Inordre:** A/B*C*D+E
- **Postordre:** AB/C*D*E+



Recorreguts vistos anteriorment

Algorithm binaryPreorder(p)

visit(p)

if p is an internal node **then**

binaryPreorder (p.left())

binaryPreorder (p.right())

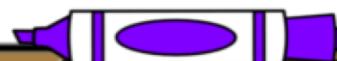
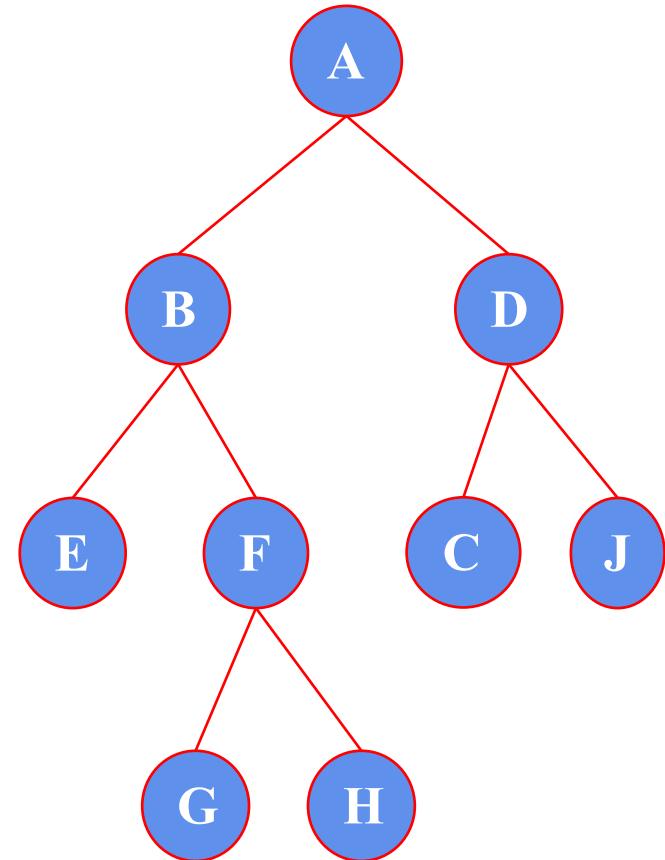
Algorithm binaryPostorder(p)

if p is an internal node **then**

binaryPostorder (p.left())

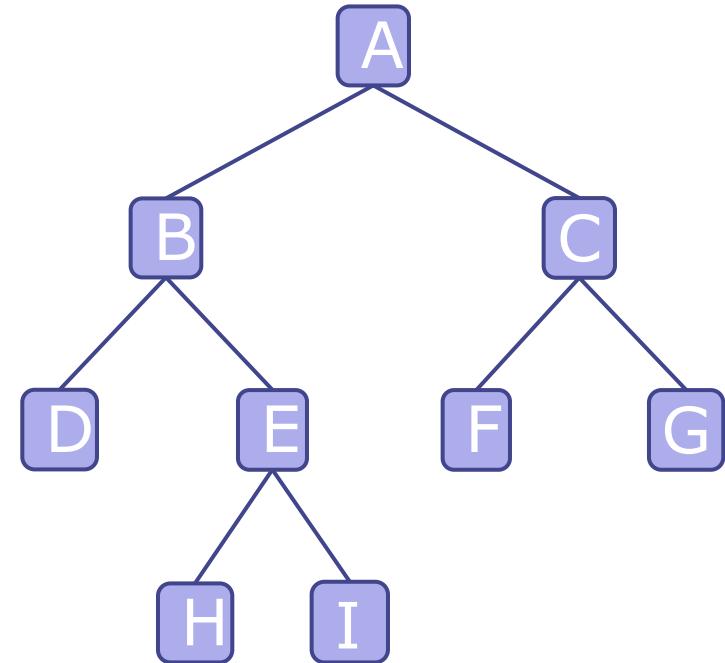
binaryPostorder (p.right())

visit(p)



Exercici de recorregut en Pre-Ordre

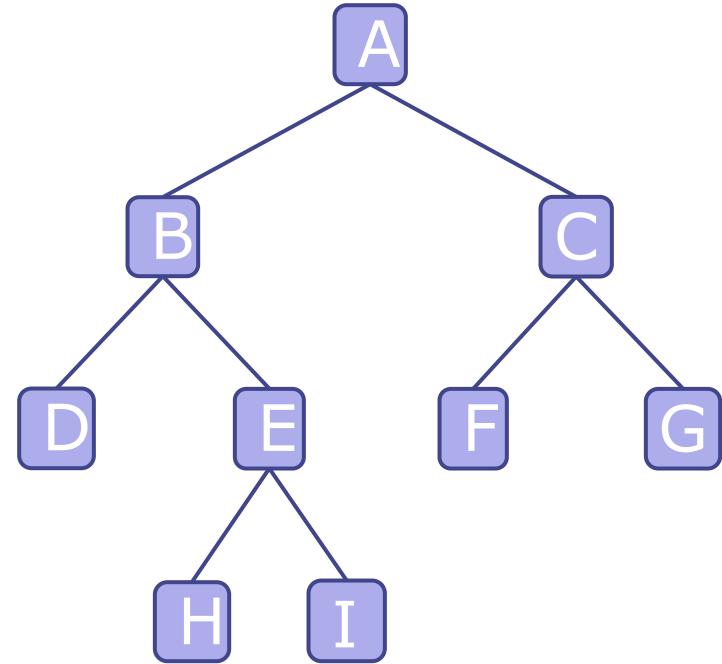
**Donat l'arbre,
escriviu el
recorregut
Pre-Ordre:**



Solució: Recorregut en Pre-Ordre

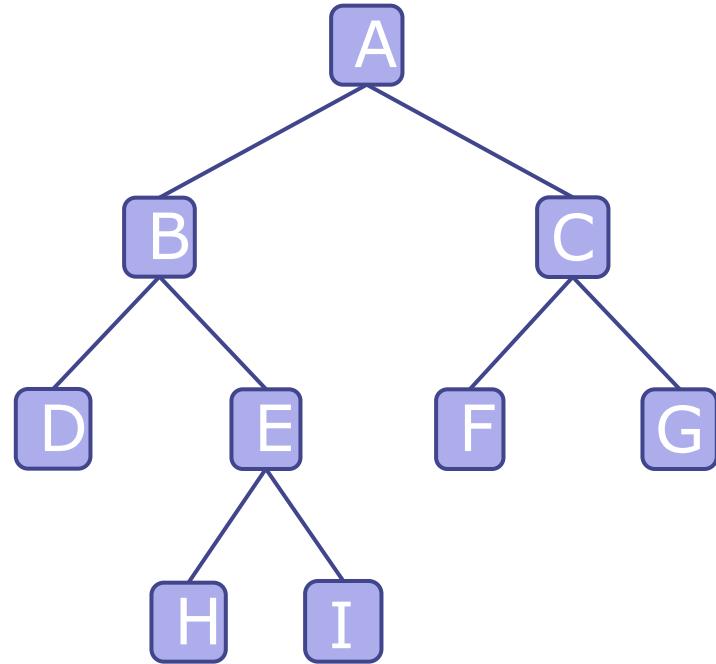
**Recorregut
Pre-Ordre:**

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



Exercici: Recorregut en Post-ordre

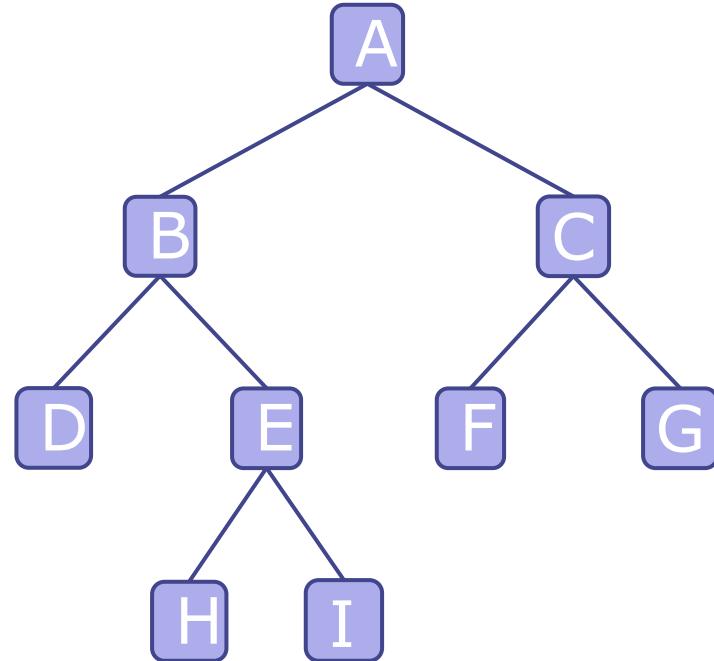
**Donat l'arbre,
escriviu el
recorregut
Post-ordre:**





Solució: Recorregut en Post-ordre

**Recorregut
Post-Ordre:**
D,H,I,E,B,F,G,C,A

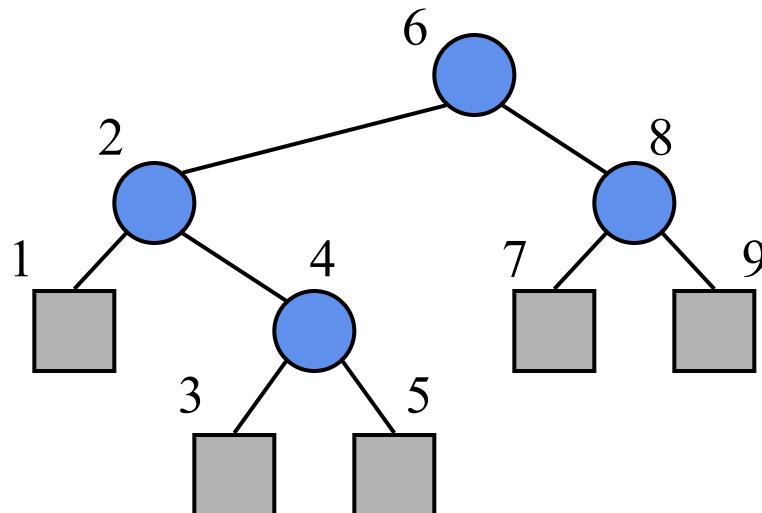


Recorregut en inordre

- En un **recorregut en inordre** un node es visita quan ja s'ha visitat el seu subarbre esquerra i abans del seu subarbre dret
- Exemple aplicació: pintar un arbre binari

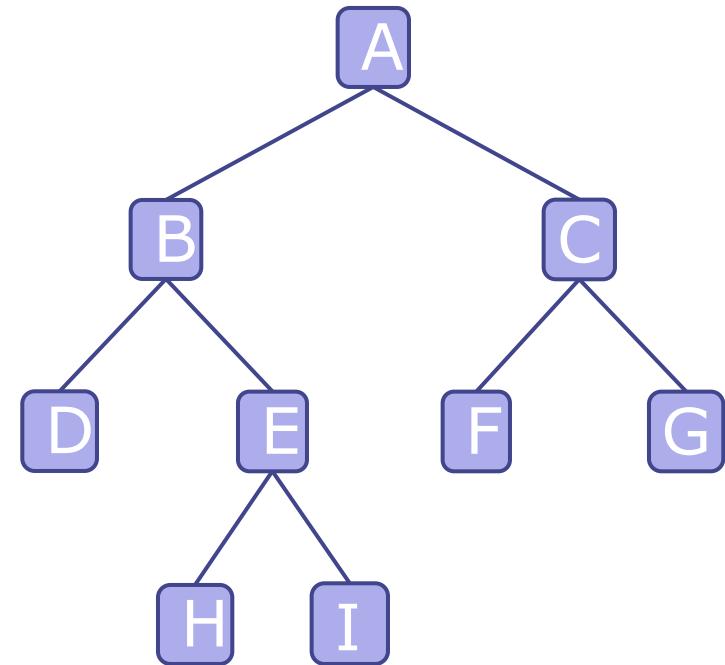
Algorithm *inOrder* (p)

```
if p is an internal node then
    inOrder(p.left())
    visit(p)
if p is an internal node then
    inOrder(p.right())
```

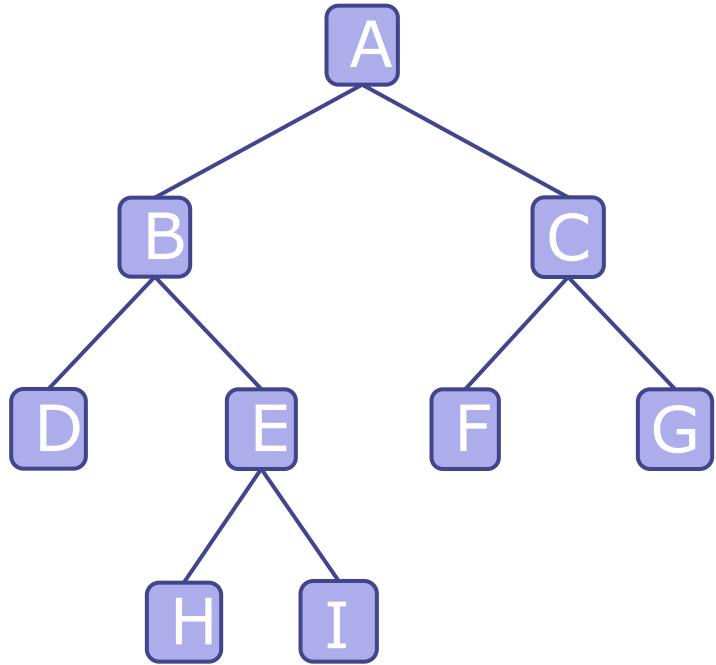


Exercici: Recorregut en In-ordre

**Donat l'arbre,
escriviu el
recorregut
In-Ordre:**



Solució: Recorregut en In-ordre

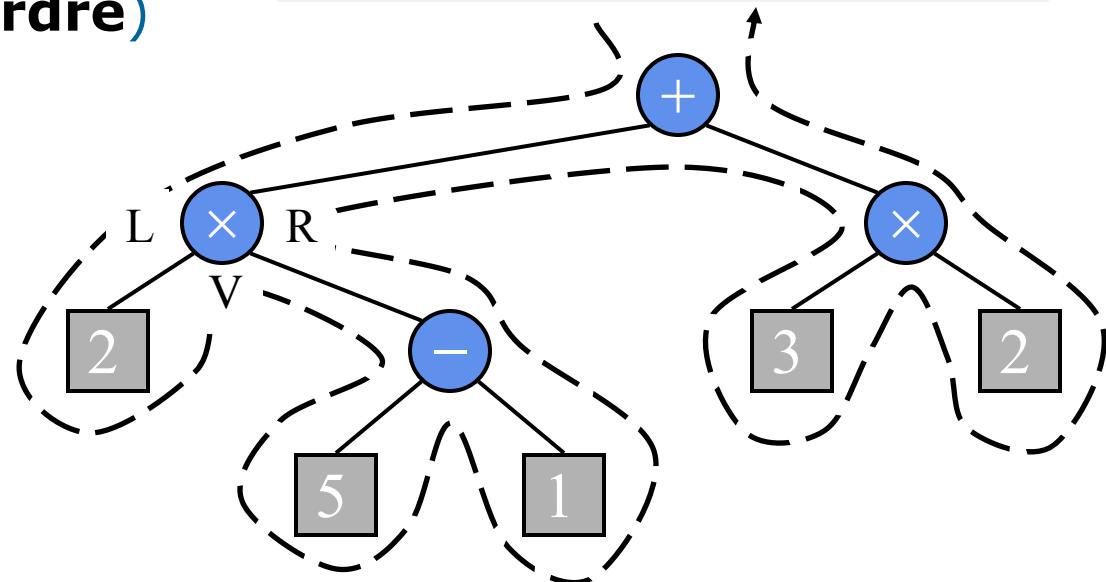


Recorregut d'Euler

- **Recorregut genèric d'un arbre binari**
- Inclou els casos especials de recorreguts: preordre, inordre i postordre
- Cada node es visita tres cops:
 - per l'esquerra (**preordre**)
 - per sota (**inordre**)
 - per la dreta (**postordre**)

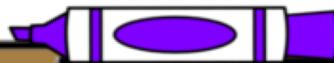
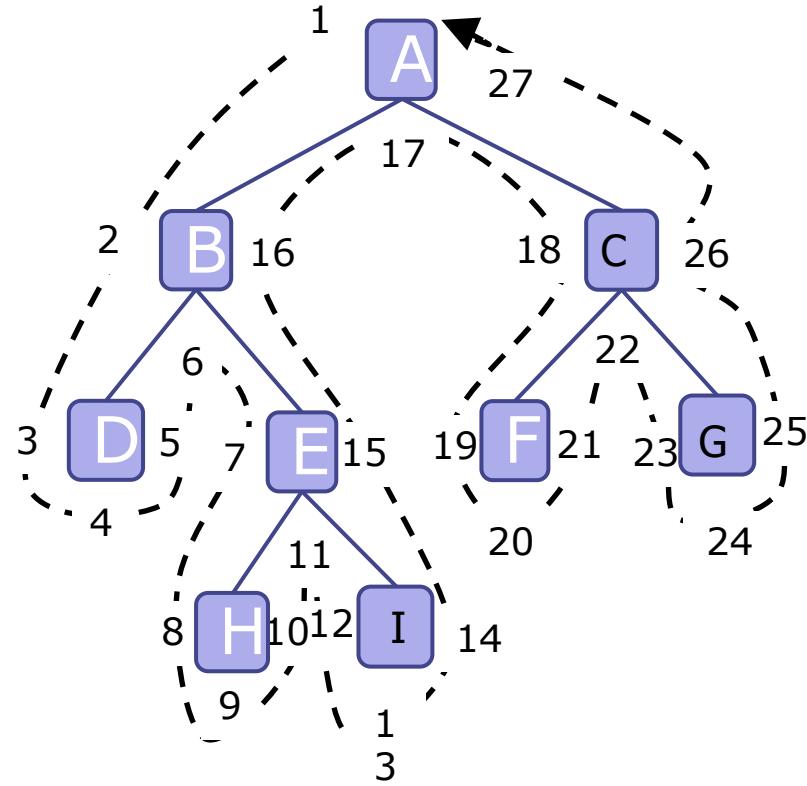


Quan un node és extern
aquestes 3 visites ocorren a
la vegada



Exercici: Escriviu el Recorregut d'Euler en Pseudo-codi

```
Algorithm eulerTour(node):
    // Input: root node of tree
    // Output: None
```



Solució Recorregut d'Euler: Pseudocodi

```
Algorithm eulerTour(node):
    // Input: root node of tree
    // Output: None

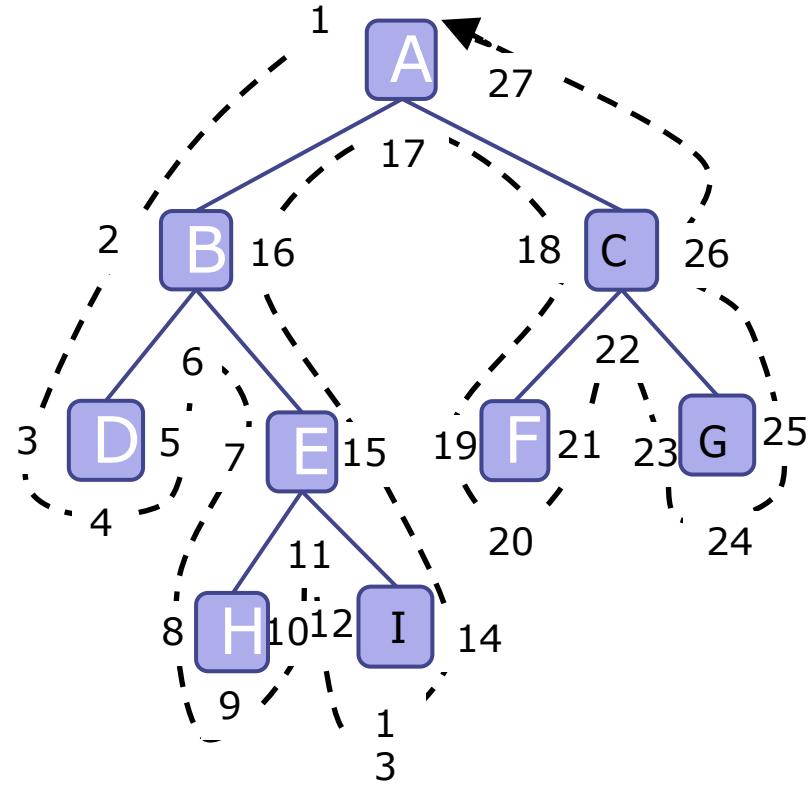
    visitLeft(node) // Pre-order

    if node has left child:
        eulerTour(node.left())

    visitBelow(node) // In-order

    if node has right child:
        eulerTour(node.right())

    visitRight(node) // Post-order
```





Quan usar cada tipus de recorregut?

- Com sabem quin és el millor mètode per recórrer un arbre?
 - Algunes vegades no importa,
 - Però normalment hi ha un mètode que ens permet solucionar el problema de forma molt més senzilla i eficient

A continuació teniu 5 problemes, decidiu quin tipus de recorregut és el que creieu més adequat per solucionar el problema



Problema 1

- Indicar en cada node quants descendents hi ha
- Millor recorregut:



Solució Problema 1

- Indicar en cada node quants descendents hi ha
- Millor recorregut: **post-ordre**
 - És fàcil calcular el nombre de descendents d'un node si ja sabem quants descendents tenen els seus fills
 - El post-ordre visita els nodes fills abans que el node pare, això és exactament el que volem



Problema 2

- Donada l'arrel d'un arbre, determinar si un arbre és perfecte.
- Millor recorregut:



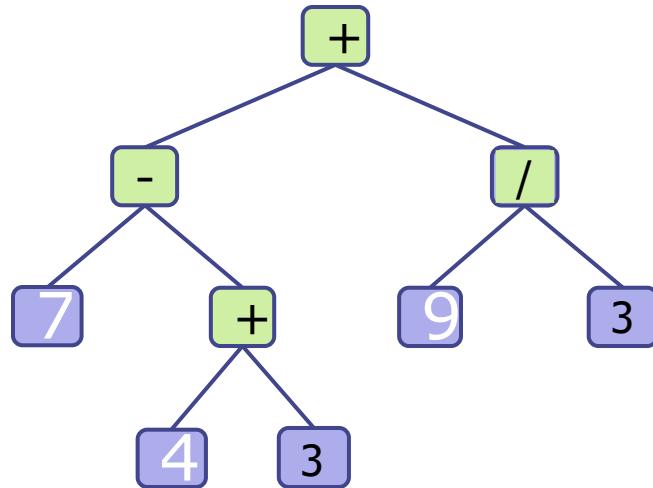
Solució Problema 2

- Donada l'arrel d'un arbre, determinar si un arbre és perfecte.
- Millor recorregut: **breadth-first**
 - La millor solució es troba explorant l'arbre nivell a nivell
 - Podem fer un seguiment del nivell actual i comptar el nombre de nodes que té
 - Cada nivell tindrà el doble de nodes que el nivell anterior

Problema 3

- Donat l'arbre que representa una expressió aritmètica, avaluar el seu resultat:

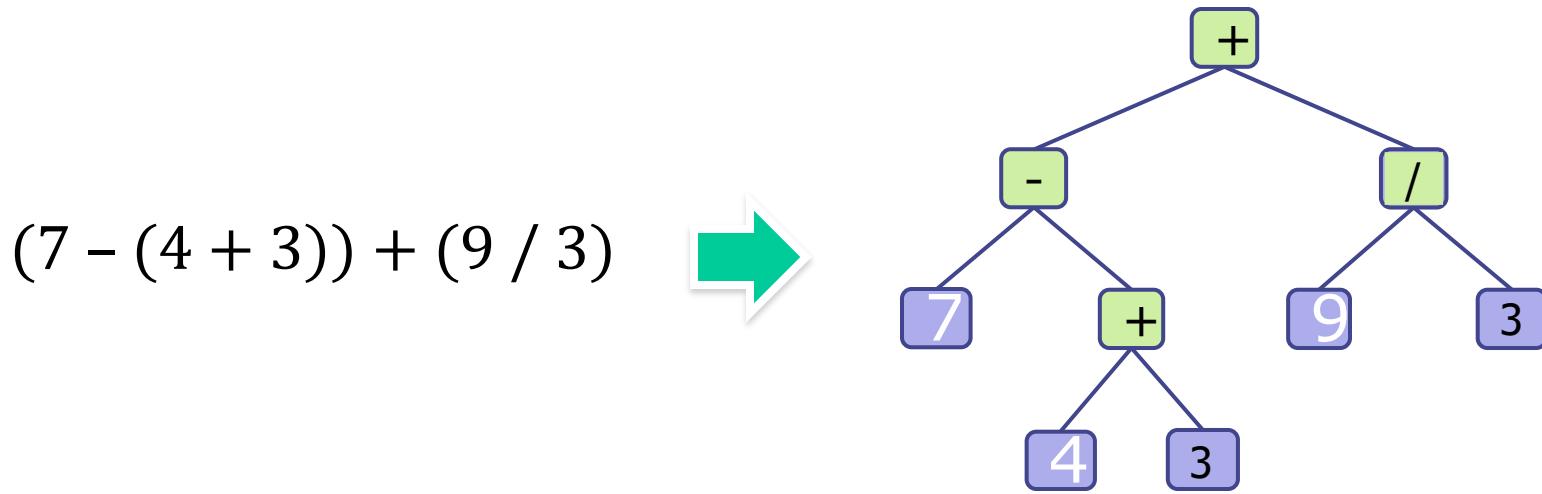
$$(7 - (4 + 3)) + (9 / 3)$$



- Millor recorregut:

Solució Problema 3

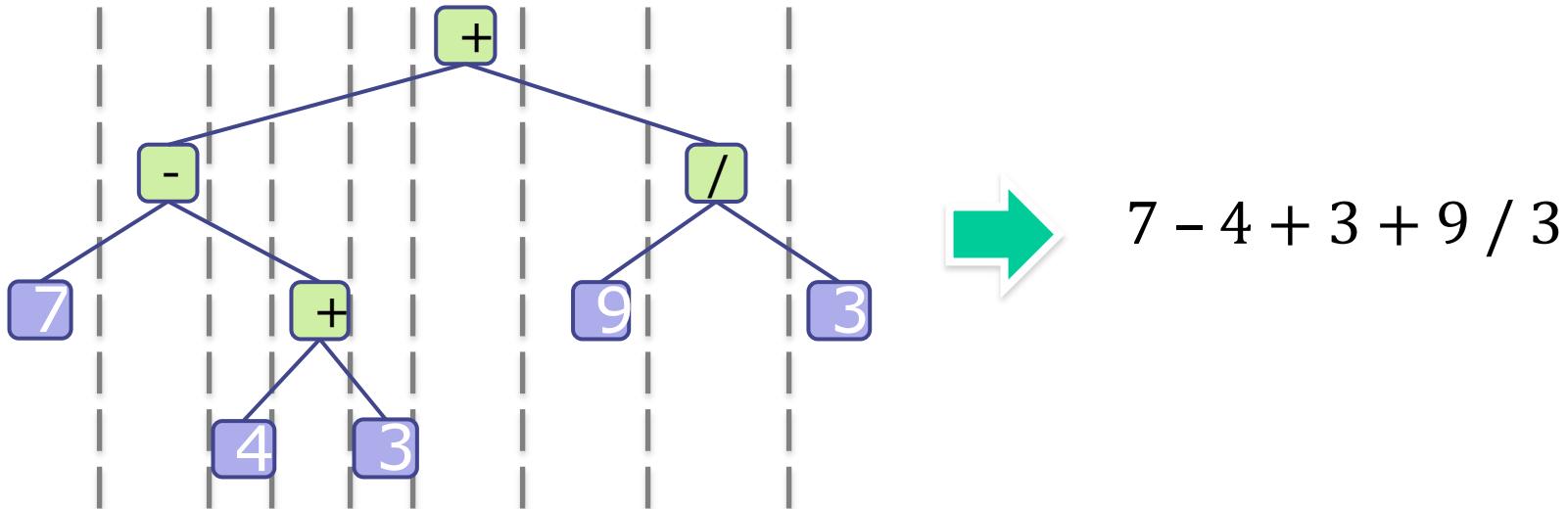
- Donat l'arbre que representa una expressió aritmètica, avaluar el seu resultat:



- Millor recorregut: **post-ordre**
 - Per avaluar l'expressió aritmètica, primer cal avaluar les sub-expressions de cada fill

Problema 4

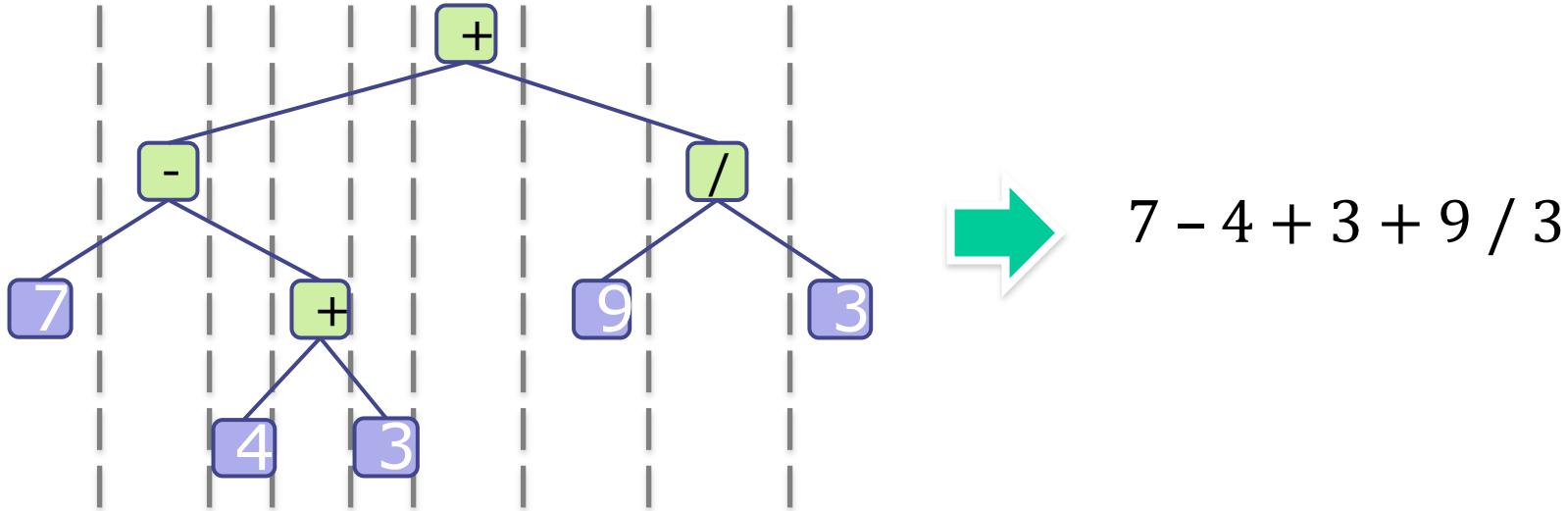
- Donada una expressió aritmètica, imprimir el seu resultat sense parèntesis



- Millor recorregut:

Solució Problema 4

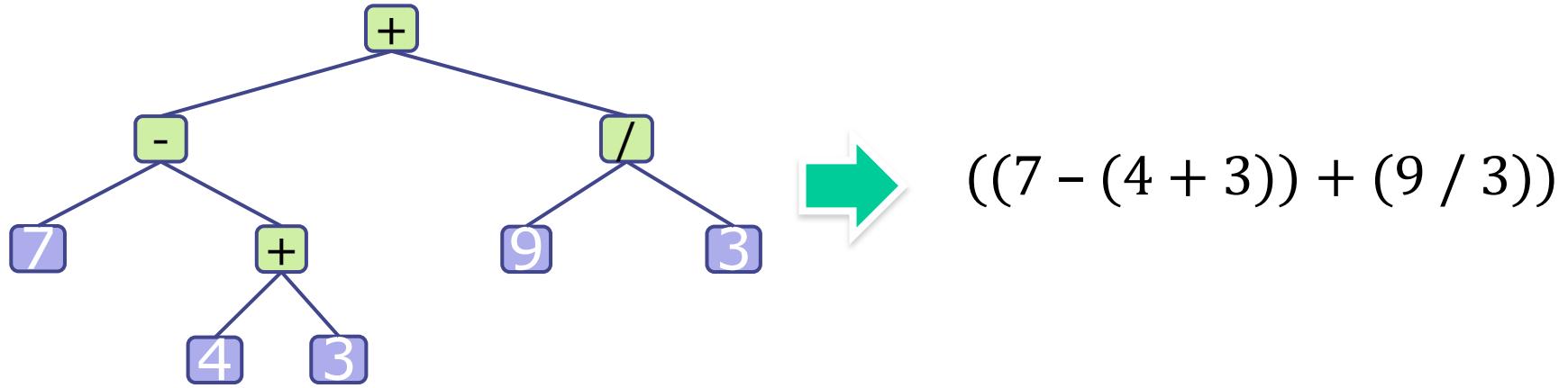
- Donada una expressió aritmètica, imprimir el seu resultat sense parèntesis



- Millor recorregut: in-ordre
 - El recorregut inordre ens dona els nodes d'esquerra a dreta

Problema 5

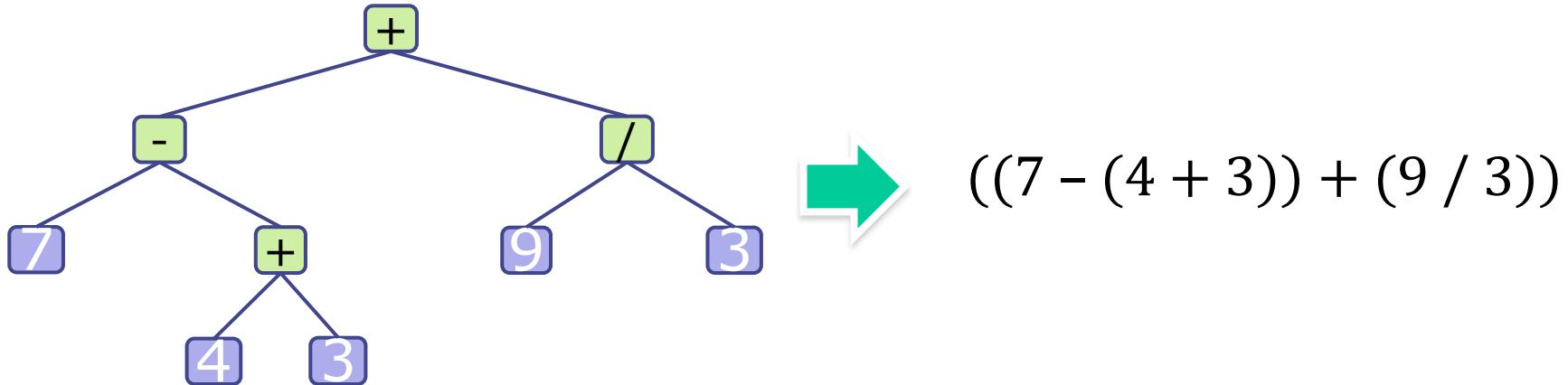
- Donat un arbre que representa una expressió aritmètica, imprimir l'expressió amb parèntesis



- Millor recorregut:

Solució Problema 5

- Donat un arbre que representa una expressió aritmètica, imprimir l'expressió amb parèntesis



- Millor recorregut: **Recorregut d'Euler**
 - Si el node és un node intern (un operador):
 - Per al pre-ordre imprimir "("
 - Per al in-ordre imprimir l'operador
 - Per al post-ordre imprimir ")"
 - Si el node és una fulla (un número)
 - No fer res per al pre-ordre ni post-ordre
 - Per al in-ordre, imprimir el número



Exercicis

Exercici 1. Comptar el número de nodes d'un arbre

Exercici 2. Comparar dos arbres, retorna CERT si són iguals, altrament retorna FALS

Exercici 3. Copiar recursivament un arbre



Tema 4 Estructures No Lineals: Arbres

Sessió Teo 9

Maria Salamó Llorente
Estructura de Dades

Grau en Enginyeria Informàtica
Facultat de Matemàtiques i Informàtica,
Universitat de Barcelona