UNIVERSITAT DE BARCELONA

Tema 4 **Estructures No Lineals: Arbres**

Sessió Teo 7

Maria Salamó Llorente

Estructura de Dades

Grau en Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,

Universitat de Barcelona

UNIVERSITAT DE BARCELONA

Contingut

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

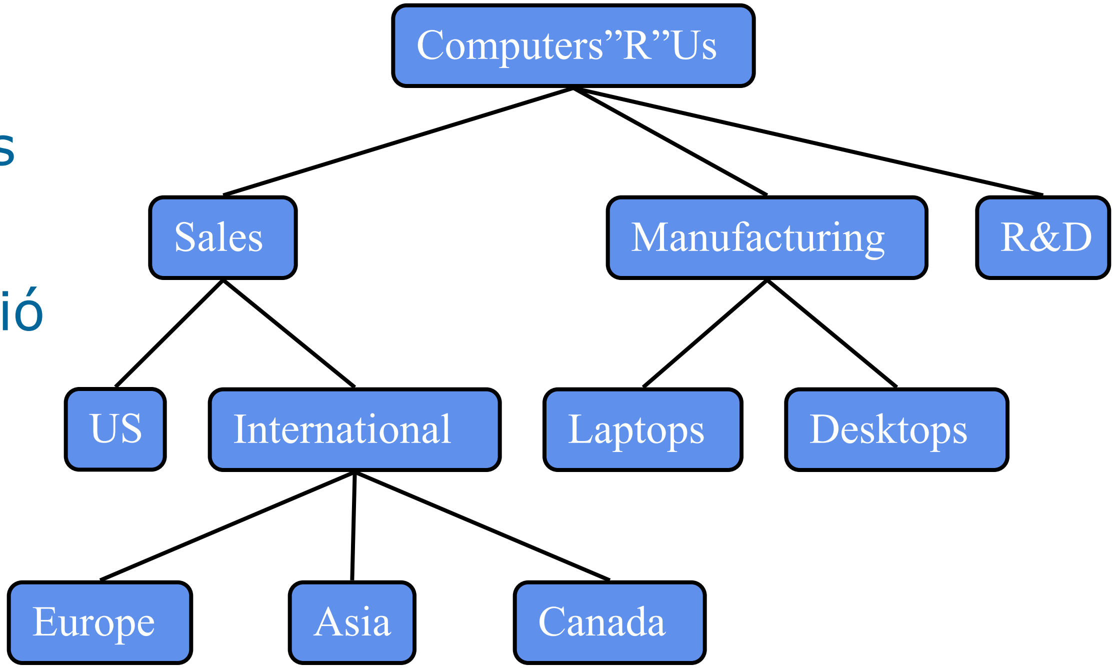
4.1 Introducció als arbres

Introducció

- Les **llistes encadenades** són estructures de dades **lineals**
  - Són seqüencials, un element darrera de l'altre
  - Cercar i recuperar informació té un cost computacional O(n)
- El **arbres**
  - Junt amb els grafs són estructures de dades **no lineals**
  - Són jeràrquics
  - Solventen els inconvenients de les llistes
  - Ofereixen diferents tipus de recorreguts
- Perquè són útils els arbres?
  - Són útils per **cercar i recuperar informació** més ràpidament que a les estructures lineals

Arbres

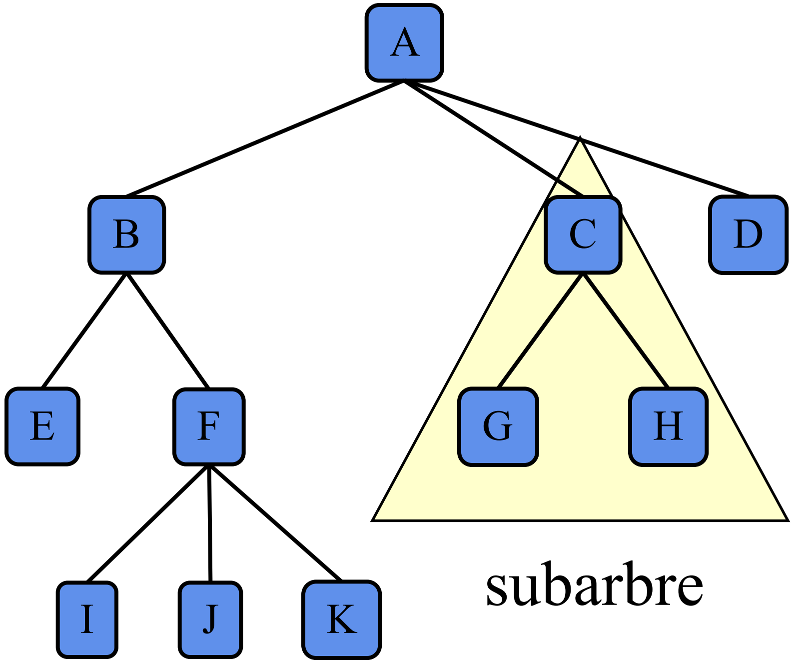
- Model abstracte d'una **estructura jeràrquica**
- Un arbre consisteix en nodes que tenen una **relació pare-fill**
- Aplicacions:
  - Organització de mapes
  - Sistemes de fitxers
  - Entorns de programació



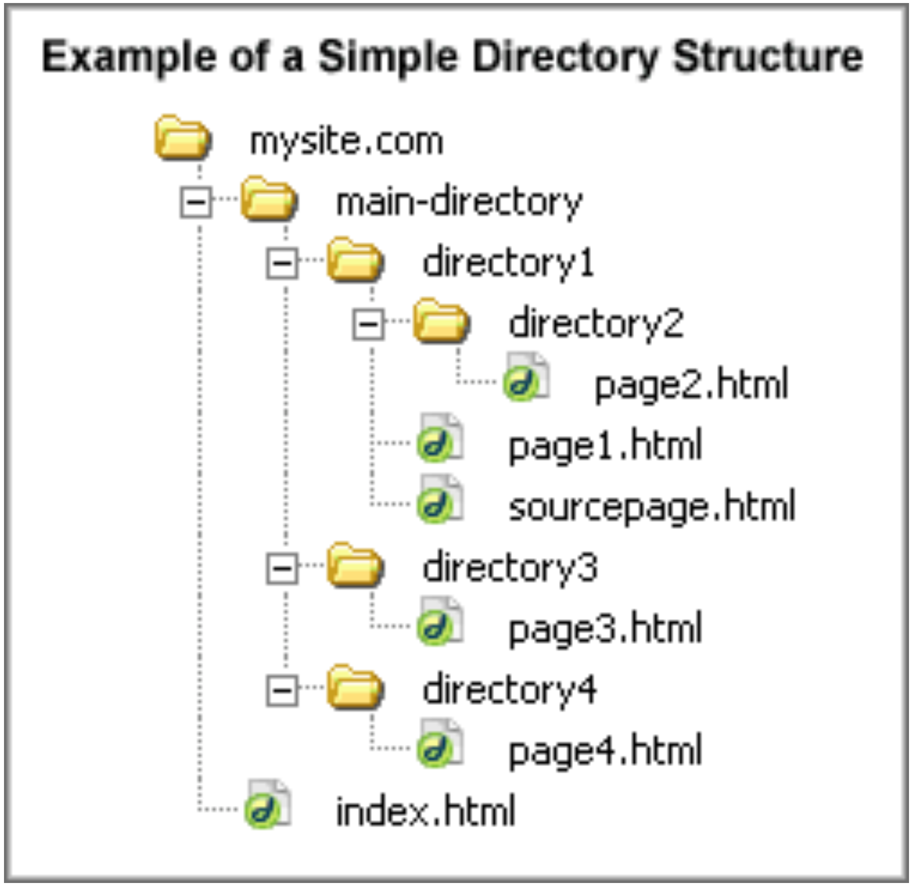
Terminologia

- **Arrel**: node sense pare (A)
- **Node intern**: node amb com a mínim un fill (A, B, C, F)
- **Node extern** (fulla): node sense fills (E, I, J, K, G, H, D)
- **Ancestres d'un node**: pare, avi, besavi, etc.
- **Profunditat d'un node**: nombre d'ancestres
- **Alçada d'un arbre**: màxima profunditat de qualsevol node (4)
- **Descendent d'un node**: fill, net, besnét, etc.

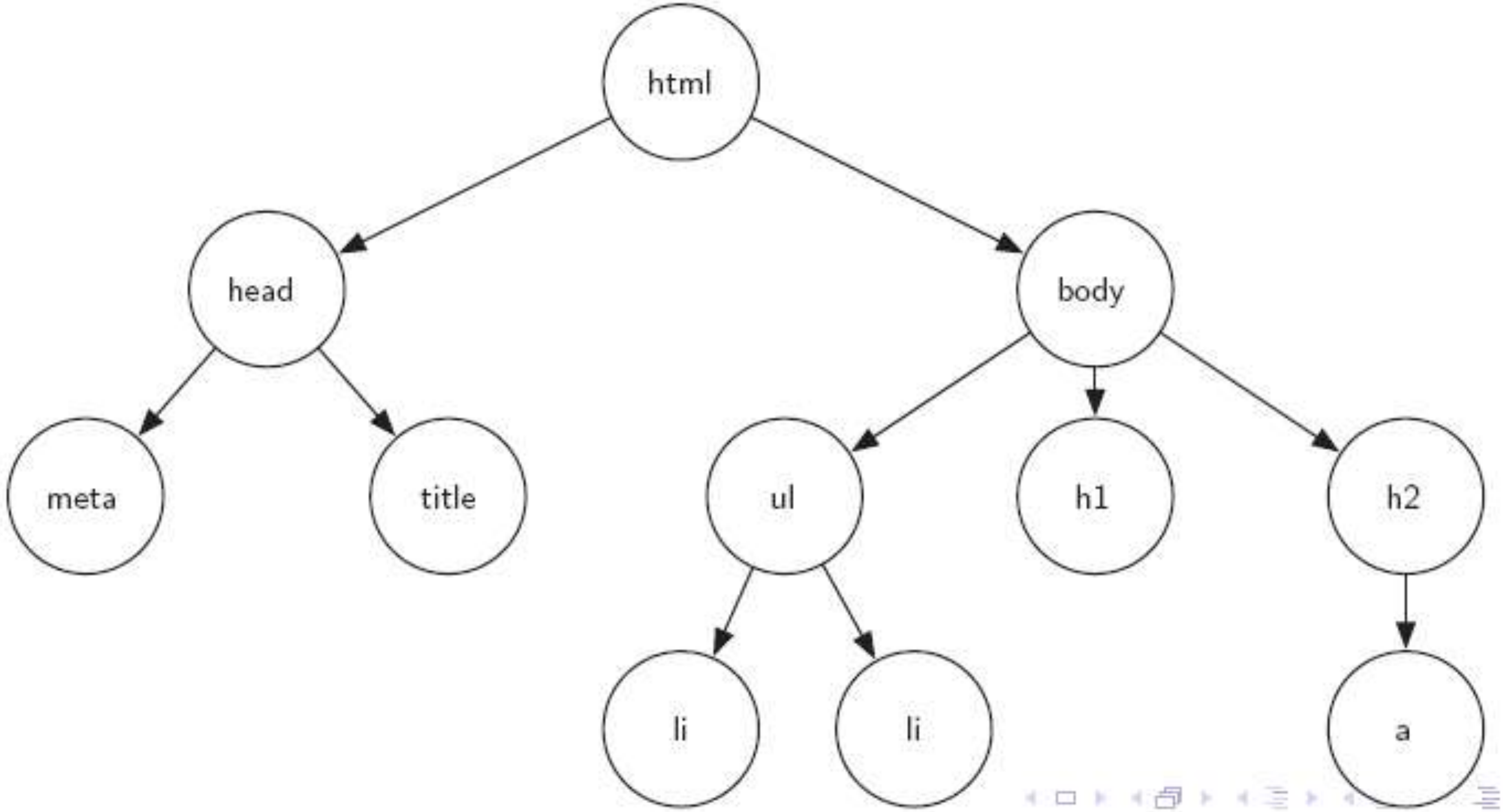
- **Subarbre**: arbre format per un node i els seus descendents



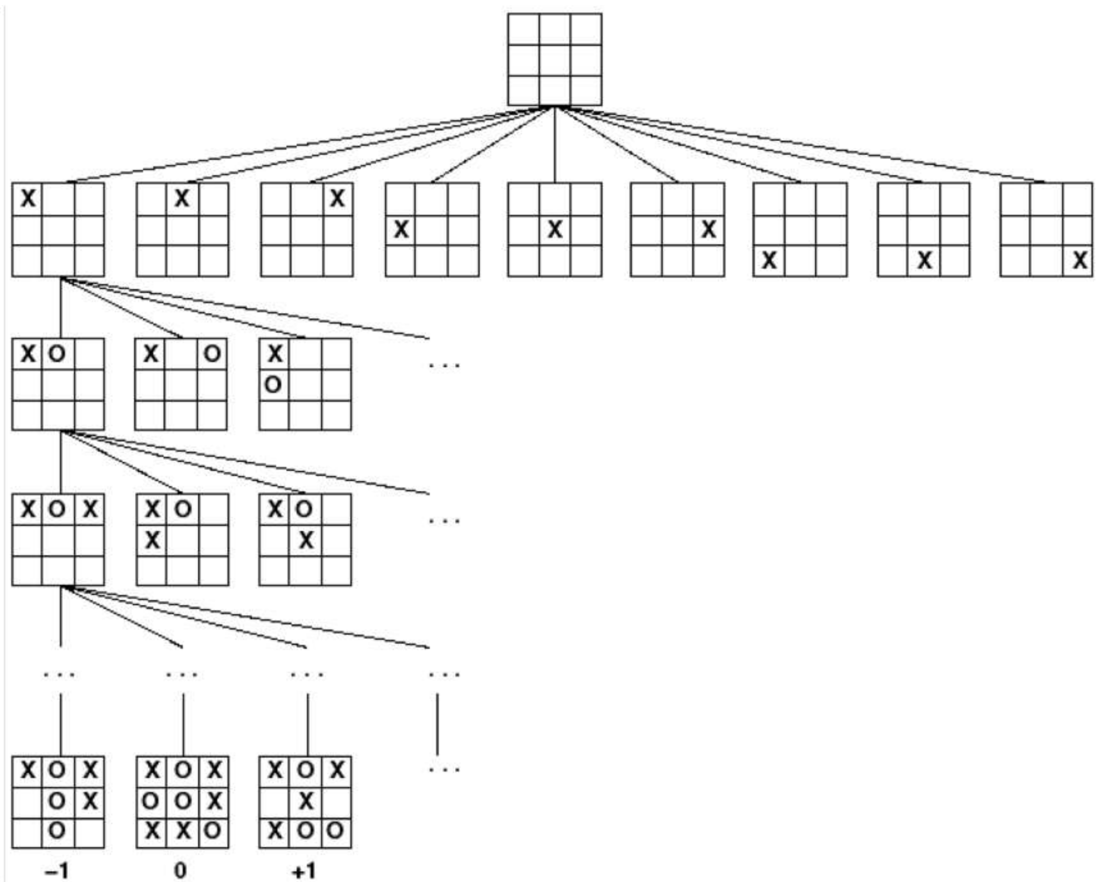
Exemple estructura de directoris i fitxers



Exemple d'arbre d'etiquetes d'una pàgina web



Exemple joc tres en ratlla

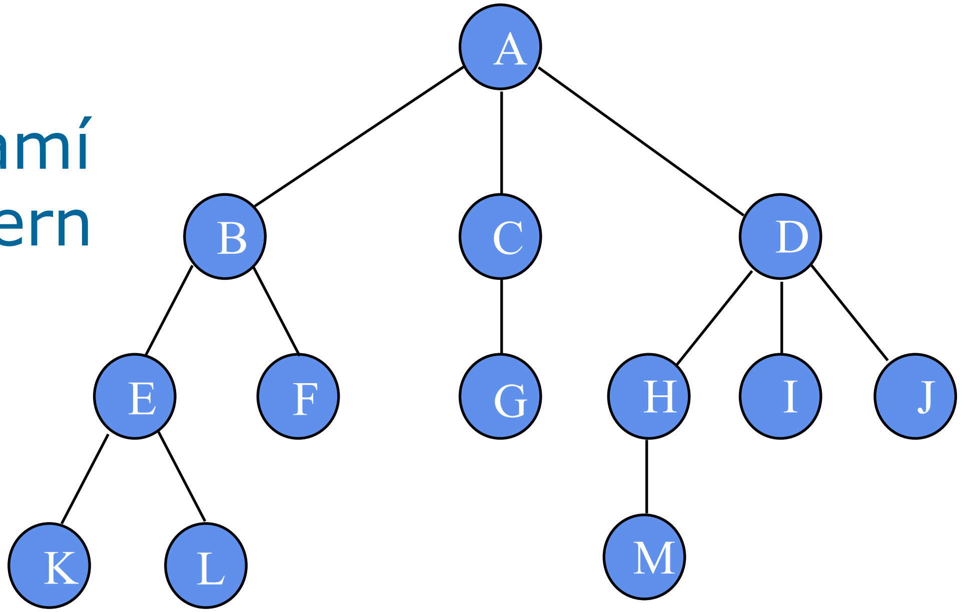


Propietats d'un arbre

- **Primera propietat**: els arbres són jeràrquics

- **Segona propietat**: Tots els fills d'un node són independents

- **Tercera propietat**: El camí fins a qualsevol node extern (fulla) és únic



TAD Arbre (Tree)

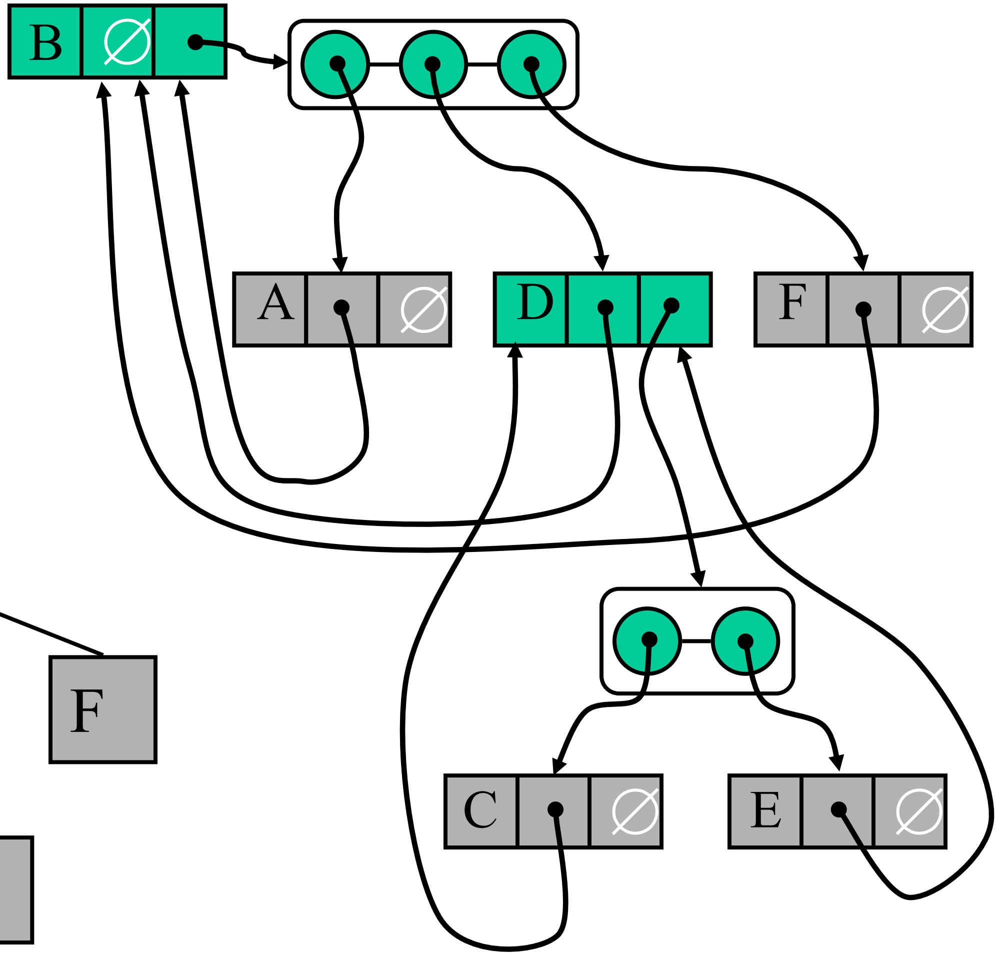
- **Mètodes genèrics**:
  - integer size()
  - boolean empty()
  - list<position> positions()
- **Mètodes d'accés**:
  - position root()
  - position p.parent()
  - list<position> p.children()

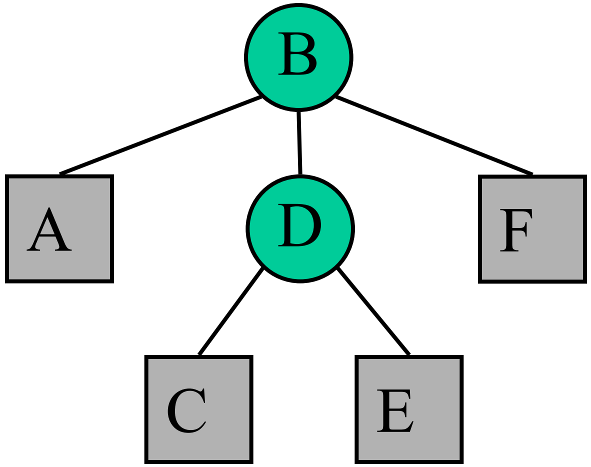
- **Mètodes de consulta**:
  - boolean p.isRoot()
  - boolean p.isExternal()
- Mètodes modificadors:
  - Es poden definir diferents mètodes de modificació segons la implementació escollida de l'arbre

Excepcions:
- Posició invàlida, Arbre buit
- Sobrepassar la frontera de l'arbre

Arbres en estructura encadenada

- Un node es representa per un objecte que guarda
  - l’Element
  - El node pare
  - La seqüència de nodes fills
- Els objectes Node implementen el TAD Position





Interfície en C++ (no està completa)

```
template <class E>
class Position<E>{
public:
  E& operator*();
  Position parent() const;
  PositionList children() const;
  bool isRoot() const;
 bool isExternal() const;
};
```

- Les posicions d'un arbre són els seus nodes
- operator* s'usa per retonar l'element que guarda el node

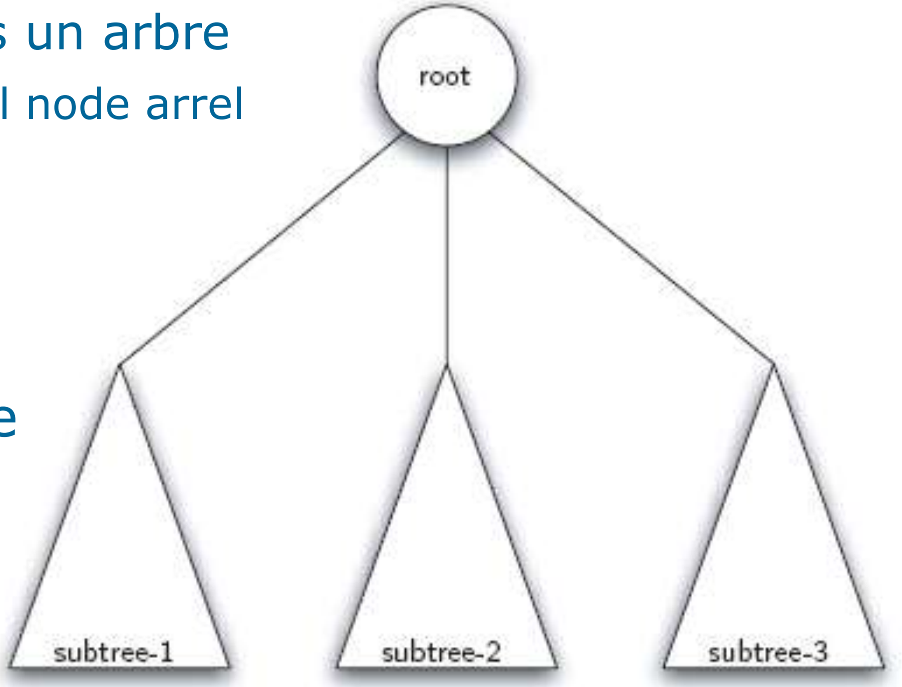
Interfície en C++ (no està completa)

```
template <class E>
class Tree<E>{
public:
  int size() const;
 bool empty() const;
  Position root() const;
  PositionList positions() const;
};
```

- PositionList segueix l'estàndard TAD llista
  - Es podria implementar amb std::list<Position>

Definició recursiva d'arbre

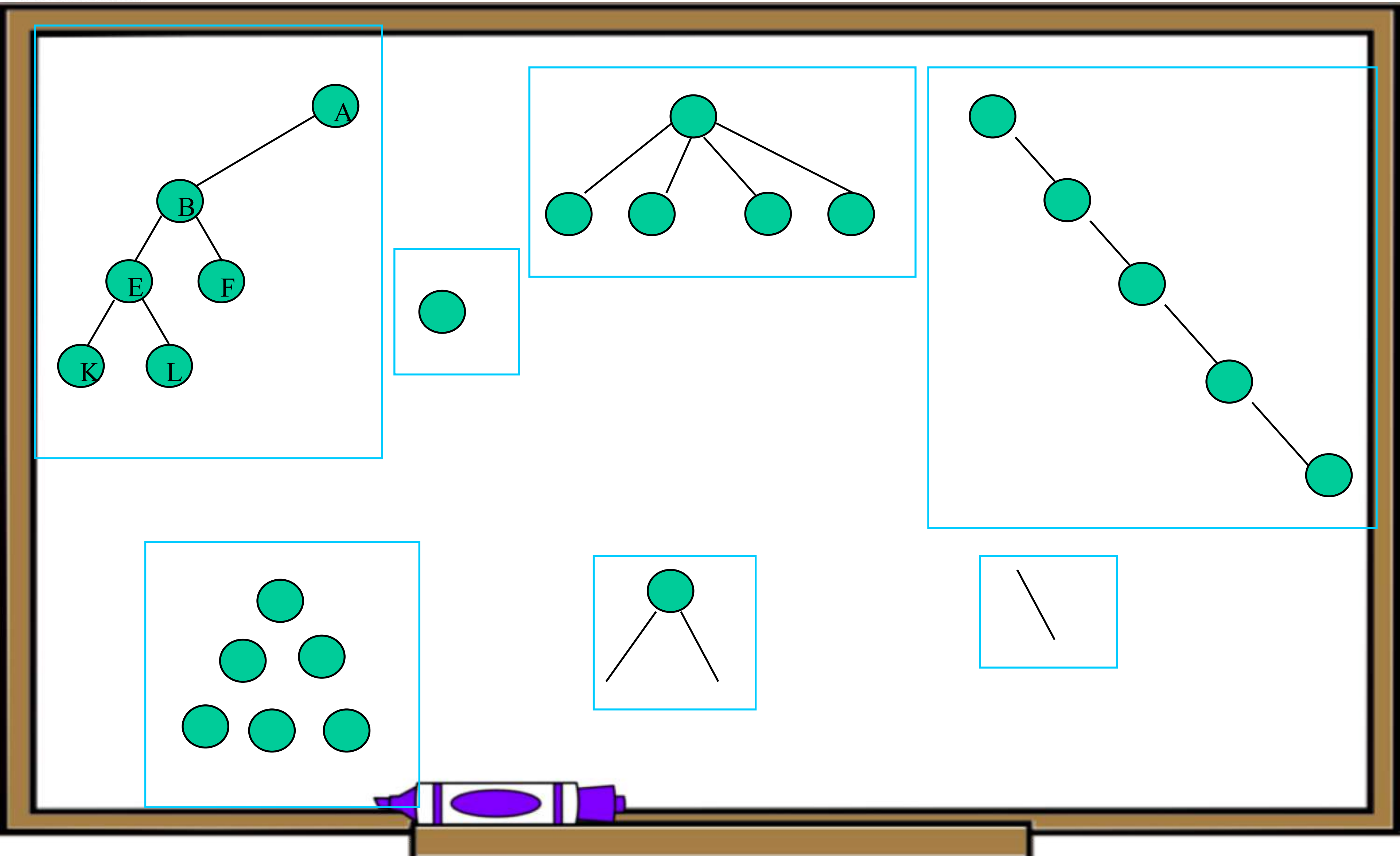
- **Definició recursiva**: l'arbre és un conjunt finit de nodes que compleix:
  - Existeix un node arrel
  - La resta de nodes estan en n (n>= 0) particions de conjunts disjunts T1, T2, …, Tn on cadascun d'aquests conjunts és un arbre
    - T1, T2, …, Tn són els subarbres del node arrel



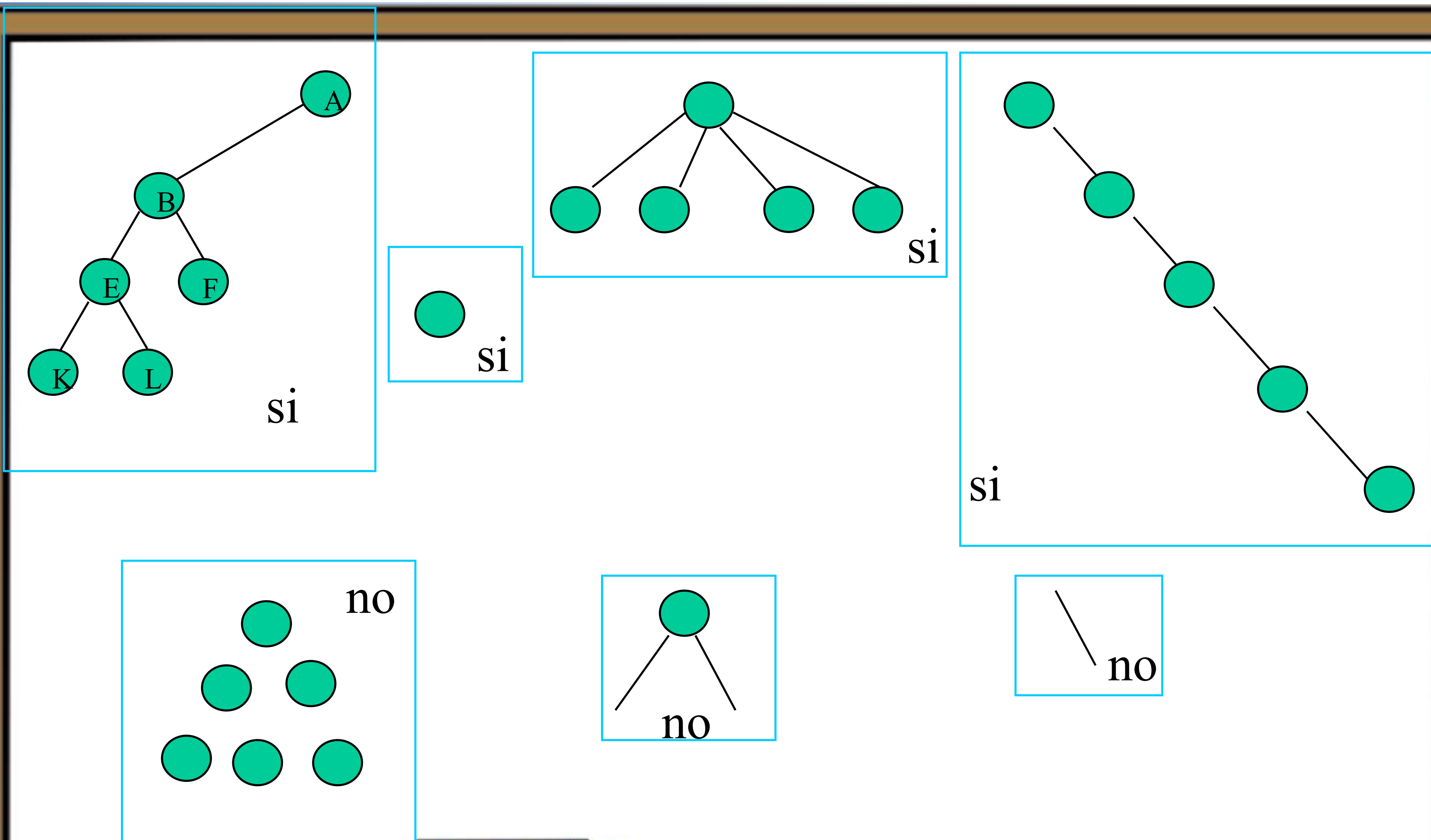
- **Funcions com**:
  - Profunditat, Alçada d'un arbre
  - Nivell d'un node,
  - Grau d'un arbre

Es calculen de forma recursiva

Són arbres?

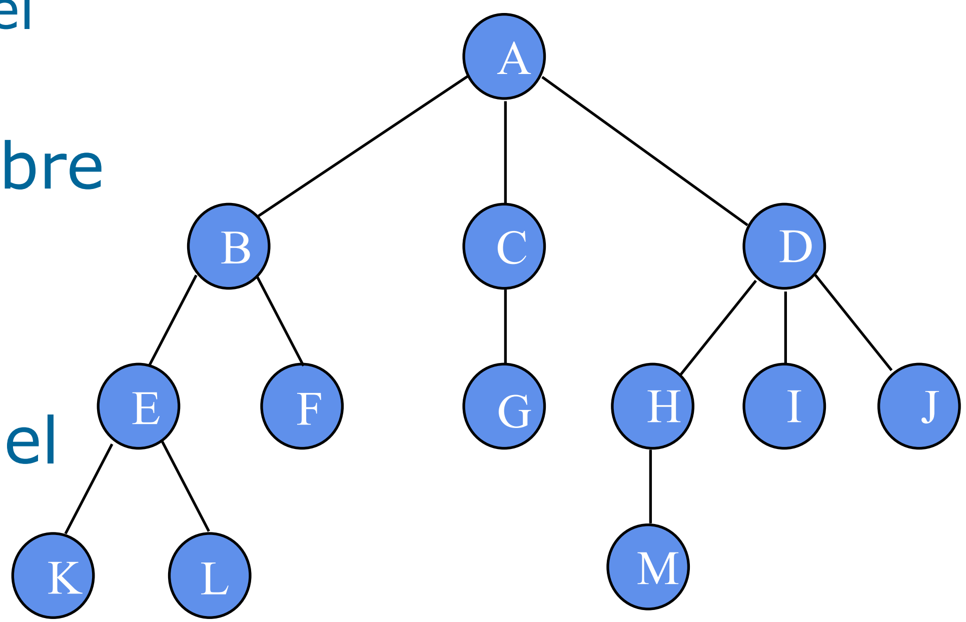


Solució: Són arbres?



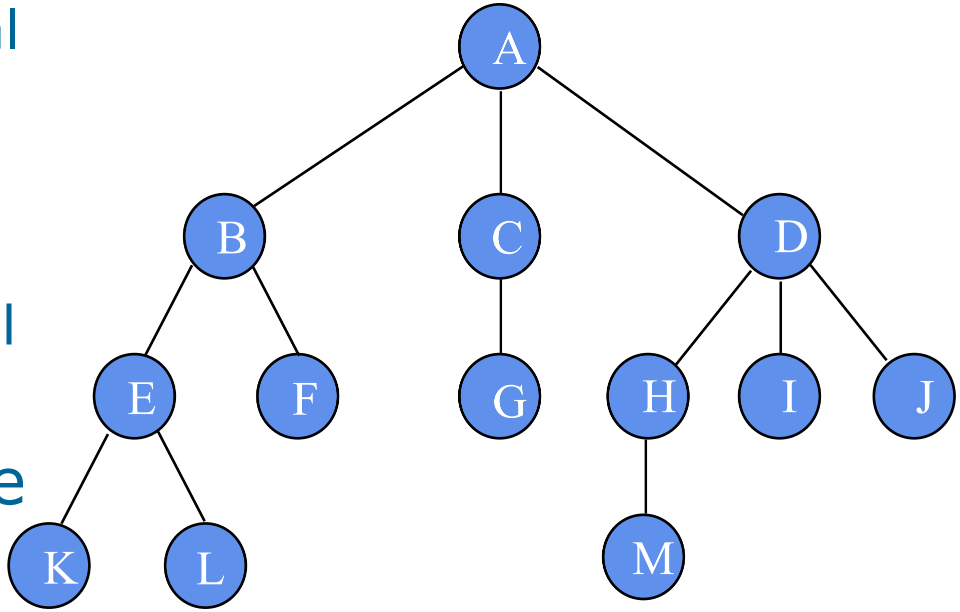
Nivell d'un node

- El **nivell** d'un node es defineix com:
  - El nivell del **node arrel és 0**
  - Si un node està en el nivell L, els seus fills estan en el nivell L+1
- La **profunditat** d'un arbre és el màxim nivell
- L'**alçada** d'un arbre és el màxim nivell + 1



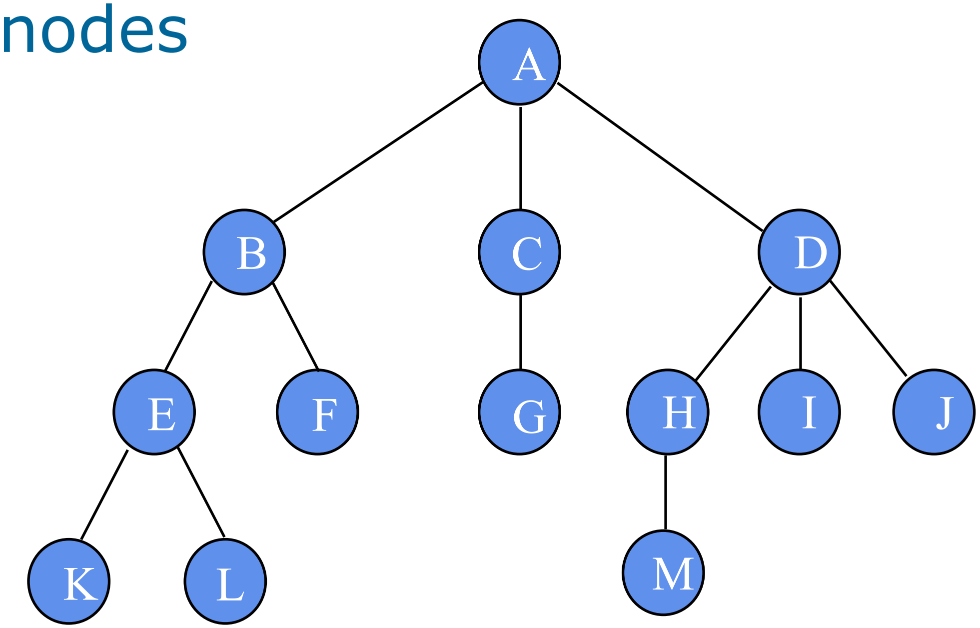
Nivell d'un node

- En aquest cas l'alçada de l'arbre és 4
- L'arbre té 3 nivells
  - El node A està al nivell 0
  - Els nodes B, C, D estan al nivell 1
  - Els nodes E, F, G, H, I, J estan al nivell 2
  - Els nodes K, L, M estan al nivell 3
- La profunditat de l'arbre és de 3



Com es mesura el grau d'un arbre

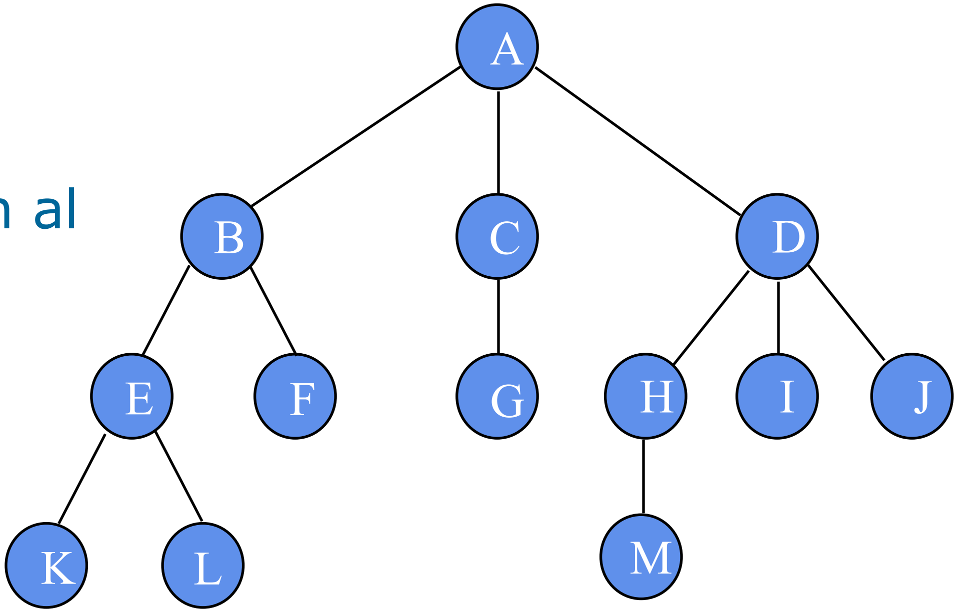
- El nombre de fills d'un node és el grau del node
  - Les **fulles o nodes externs** tenen grau zero
- **Grau d'un arbre**: és el màxim grau dels seus nodes



Com es mesura el grau d'un arbre

- El grau d'un node del B és 2
  - El node B i el E tenen grau de 2
  - El node D té un grau de 3
  - El node C i H tenen grau de 1

- **Grau de l'arbre**: 3
  - Ja que el màxim grau de qualsevol node correspon al node D amb 3 fills

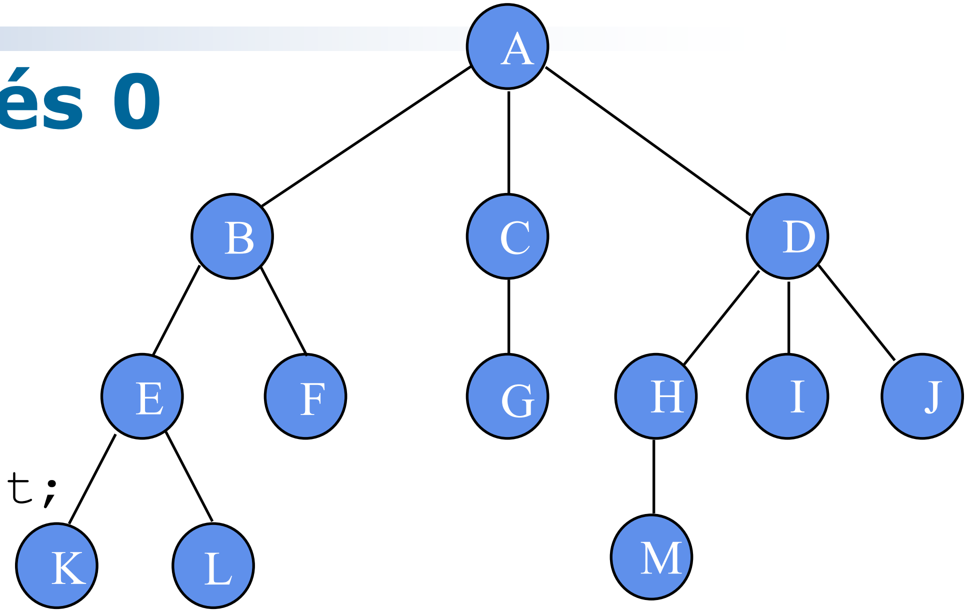


Profunditat d'un node

```
int depth(const Position & p)
{

// calculeu aquí la profunditat d'un node
de forma recursiva fent servir les funcions
definides a la classe Position

// Recordeu que per trobar la profunditat
d'un node cal saber quants ancestres té el
node



}
```

Profunditat d'un node

Profunditat de l'arrel és 0




```
template <class E>
class Position<E>{
public:
  E& operator*();
  Position parent() const;
  PositionList children() const;
  bool isRoot() const;
 bool isExternal() const;
};
```

```
template <class E>
class Tree<E>{
public:
  int size() const;
 bool empty() const;
  Position root() const;
  PositionList positions() const;
private: Position<E> *root;
};
```

Solució: Profunditat d'un node

```
int depth(const Position & p)
{
  if (p.isRoot())
    return 0;
  else
    return 1 + depth(p.parent());
}
```

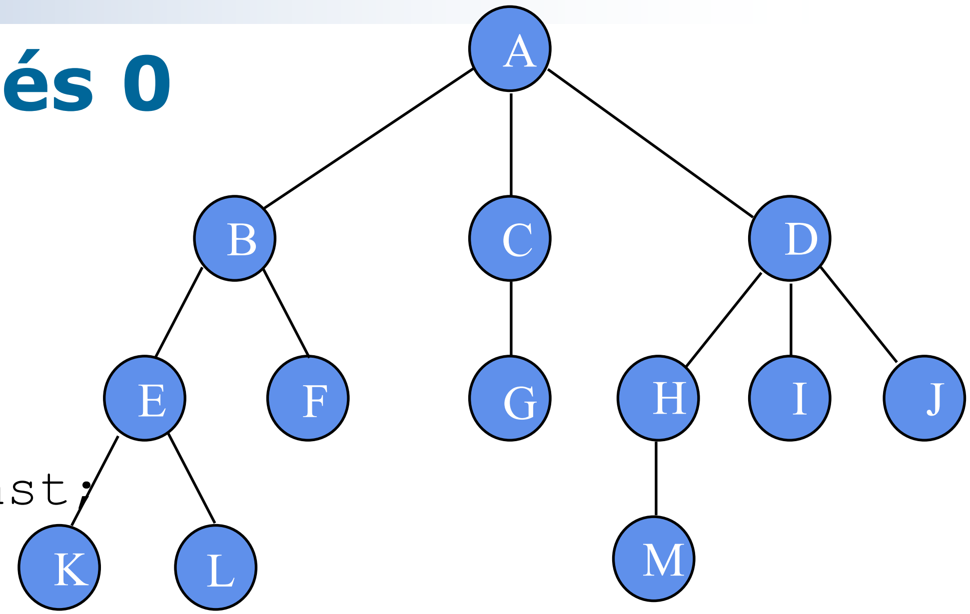
Alçada 1

```
int height1(const Tree& T)
{

// calculeu aquí l'alçada de l'arbre de
forma recursiva fent servir les funcions
definides a la classe Position i a Tree

// Una manera de calcular-ho pot ser mirant
quins nodes de l'arbre són externs i veure
quina és la profunditat d'aquests nodes,
l'alçada de l'arbre serà la màxima
profunditat dels seus nodes +1
}
```

Height 1

Profunditat de l'arrel és 0

```
template <class E>
class Position<E>{
public:
  E& operator*();
  Position parent() const;
  PositionList children() const;
  bool isRoot() const;
 bool isExternal() const;
};
```




```
template <class E>
class Tree<E>{
public:
  int size() const;
 bool empty() const;
  Position root() const;
  PositionList positions() const;
private: Position<E> *root;
};
```

Solució: Alçada 1

```
int height1(const Tree& T)
{
  int h = 0;
  PositionList nodes = T.positions();
  for (Iterator q= nodes.begin();
          q!= nodes.end(); ++q)
  {
     if (q->isExternal())
        h = max(h, depth (*q));
  }
  return h+1;
}
```

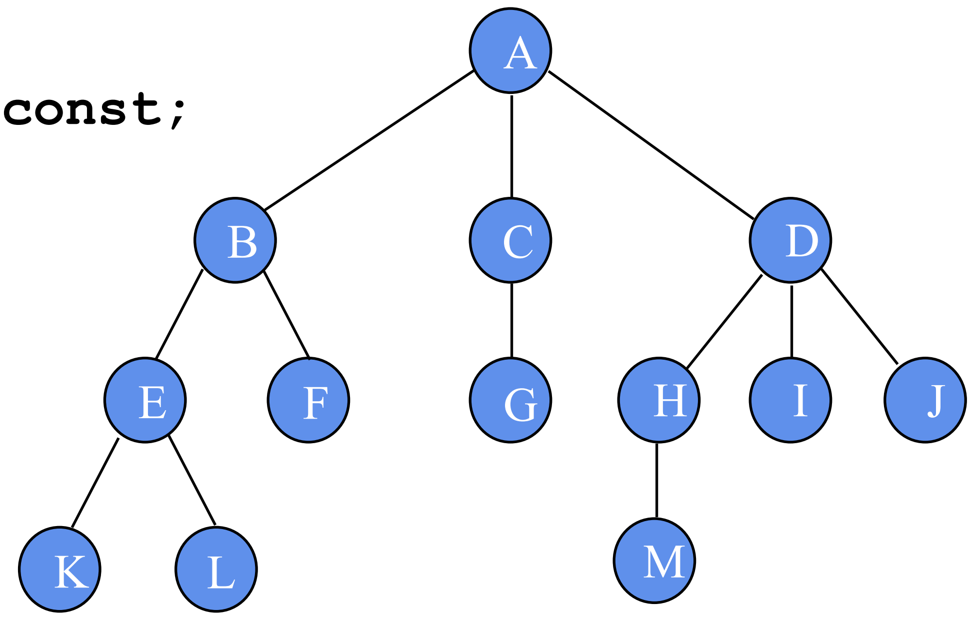
Alçada 2

```
int height2(const Tree& T, const Position& p)
{
// calculeu aquí l'alçada de l'arbre de forma
recursiva fent servir les funcions definides a la
classe Position. Inicialment a la funció se li passa
la Position del node arrel.

// Una altra manera de calcular-ho pot ser mirant
quina és l'alçada de cada node. Des del node arrel a
les fulles. Recordeu que l'alçada d'una fulla és 1.

}
```

Height 2

Profunditat o alçada de l'arrel és 0

```
template <class E>
class Position<E>{
public:
  E& operator*();
  Position parent() const;
  PositionList children() const;
  bool isRoot() const;
 bool isExternal() const;
};
```



Solució: Alçada 2

```
int height2(const Position& p)
{
  if (p.isExternal()) return 0;
  int h = 1;
  PositionList ch = p.children();
  for (Iterator q= ch.begin();
       q!= ch.end(); ++q)
  {
     h = max(h, height2(*q));
  }
 return 1+h;
}
```

Recorreguts en arbres

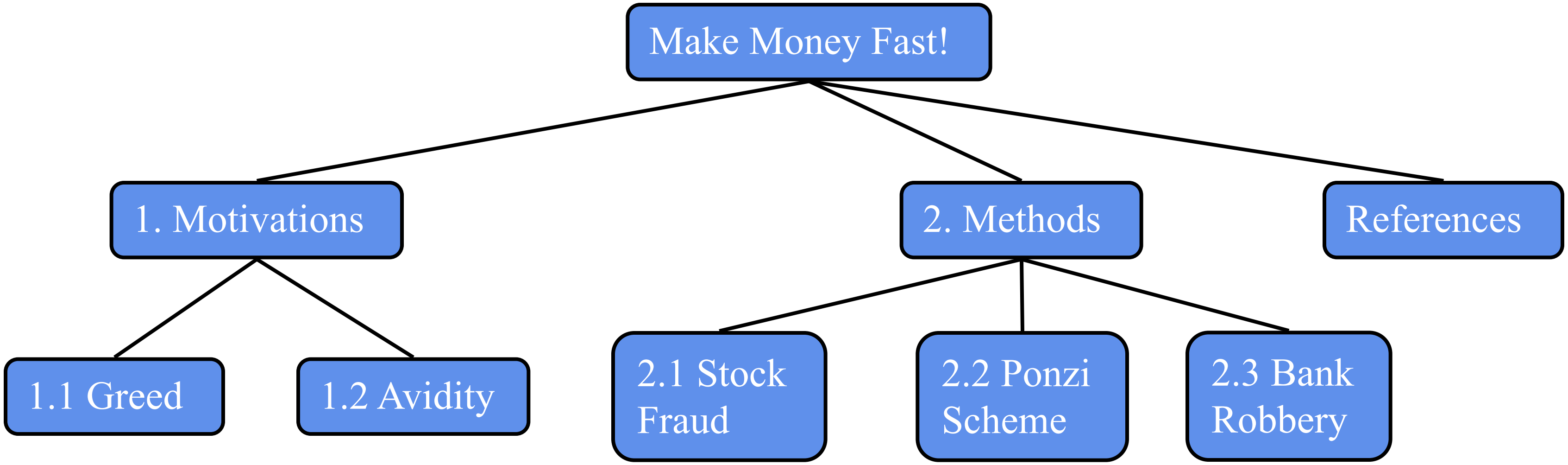
- **Un recorregut visita els nodes d'un arbre d'una manera sistemàtica**

- Les dues maneres més habituals són:
  - **Recorregut en preordre**: un node es visita abans que els seus descendents
  - **Recorregut en postordre**: Un node es visita després dels seus nodes descendents

- En arbres binaris veurem més tipus de recorreguts.

Recorreguts en arbres

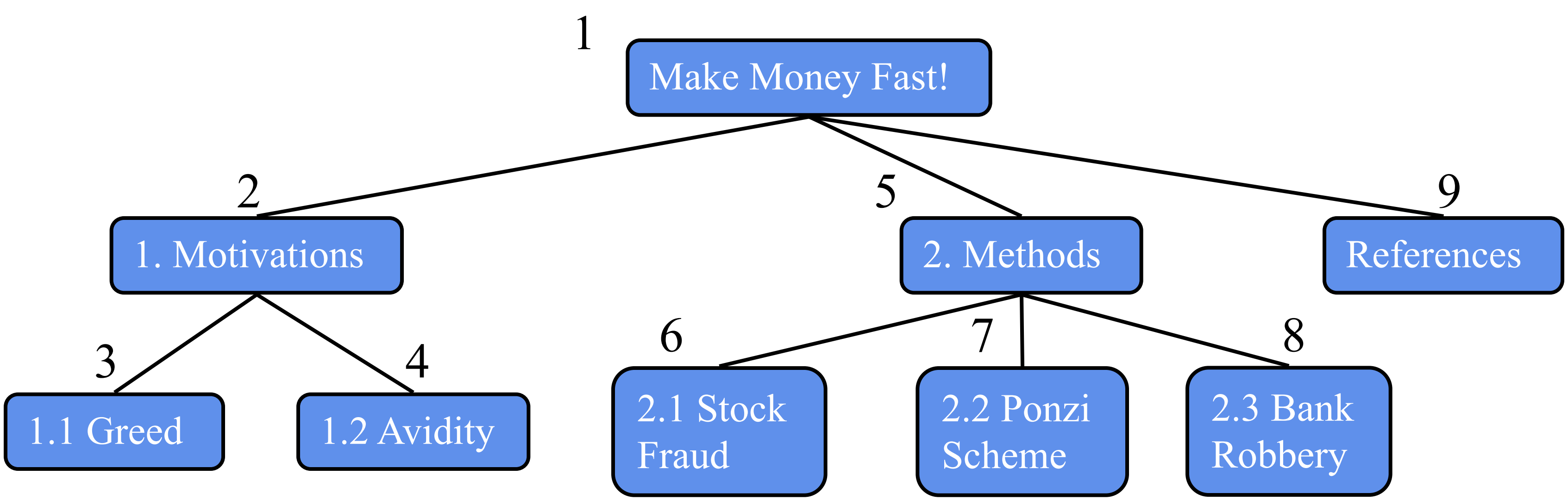
- Exemple aplicació:
  - Imprimir la taula de continguts d'un document estructurat



Recorregut en preordre

- En un **recorregut en preordre**, un node es visita abans que els seus descendents
- Exemple aplicació: imprimir la taula de continguts d'un document estructurat

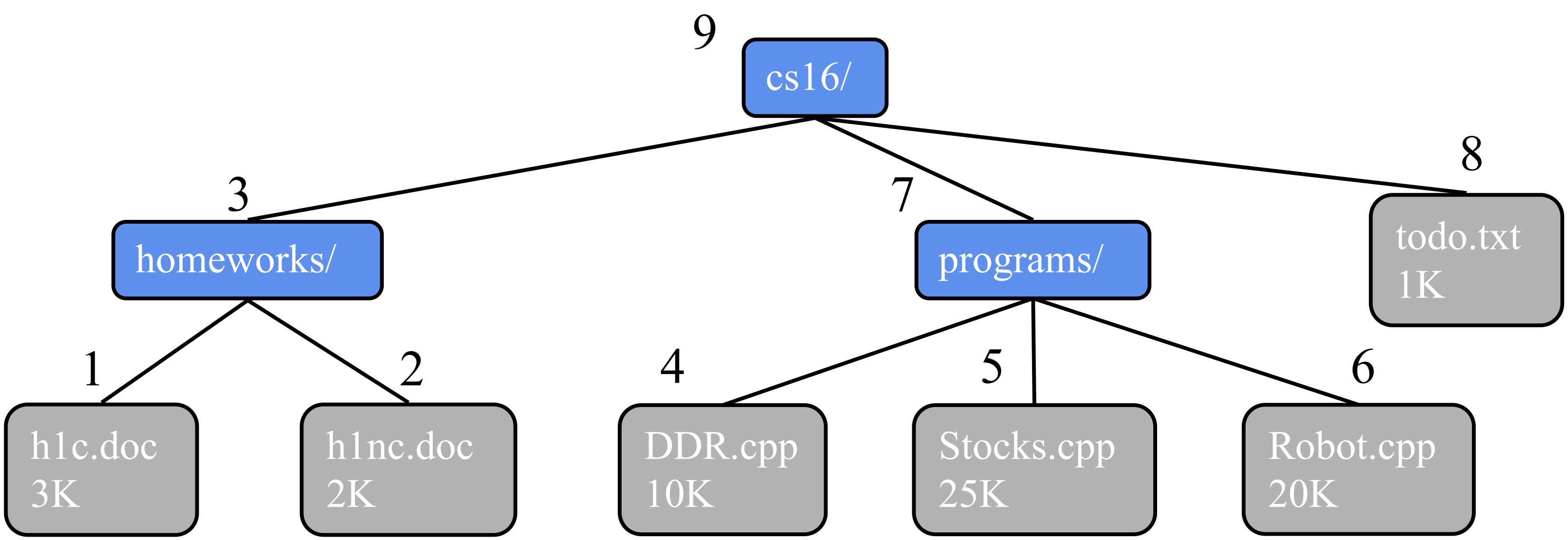
Algorithm ***preOrder***(*v*)
    visit(*v*)
    for each child *w* of *v*
        preOrder (*w*)



Recorregut en postordre

- En un **recorregut en postordre**: Un node es visita després dels seus nodes descendents
- Exemple aplicació: calcular l'espai usat pels fitxers en un directori i subdirectoris

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



Exercicis

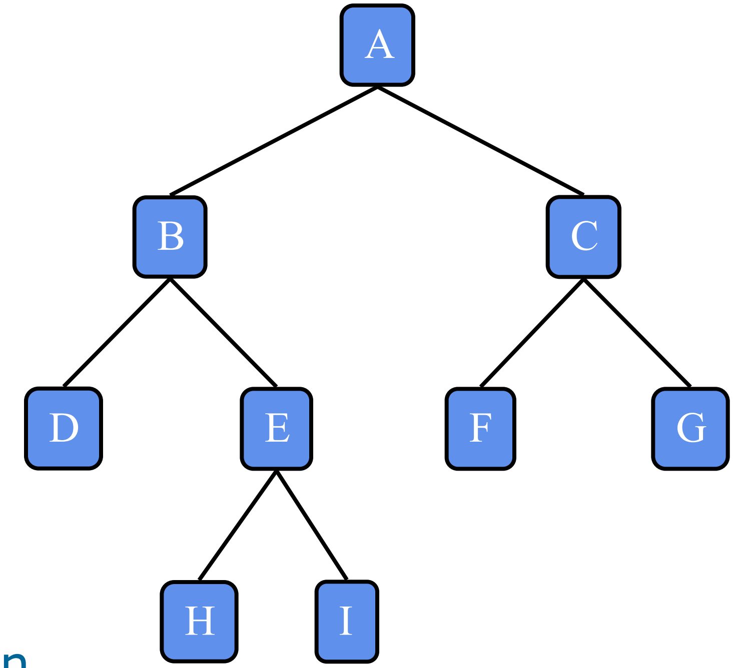
Exercici 1. Implementeu el recorregut en preordre en C++

Exercici 2. Implementeu el recorregut en postordre en C++

4.2 Arbres binaris

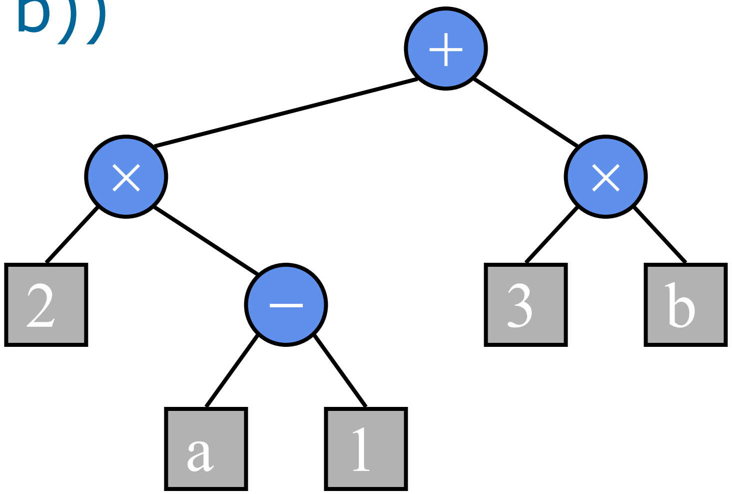
Arbres binaris

- Un arbre binari és un arbre amb les següents **propietats**:
  - Cada node intern té com a **màxim dos fills**
    - Pot ser 0, 1, o 2 fills.
  - Els fills d'un node són un parell ordenat
- Els fills d'un node intern s'anomenen **fill esquerra** i **fill dret**
- **Exemples d'ús**:
  - Expressions aritmètiques
  - Processos de decisió (s'anomenen arbres de decisió)
  - Cerques



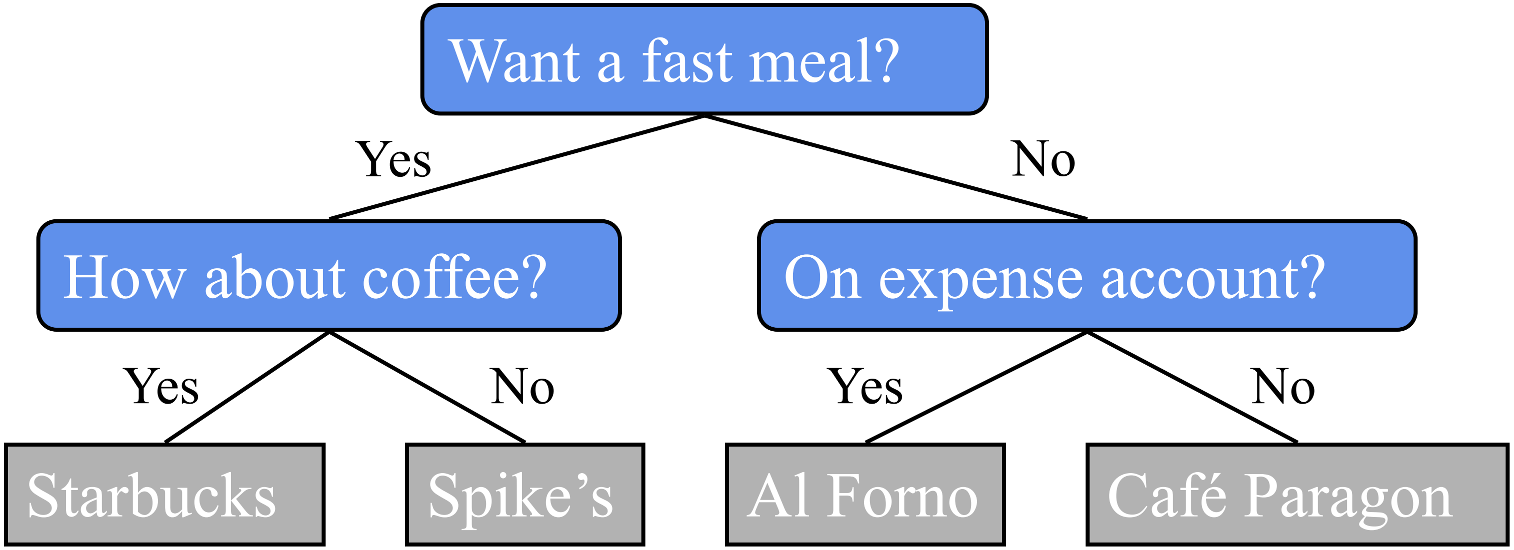
Arbre d'expressions aritmètiques

- Es pot associar un arbre binari a una expressió aritmètica
  - nodes interns: operadors
  - nodes externs: operands
- Fent un recorregut de l'arbre es pot fer l'avaluació de l'expressió aritmètica
- Exemple: $(2 \times (a - 1) + (3 \times b))$



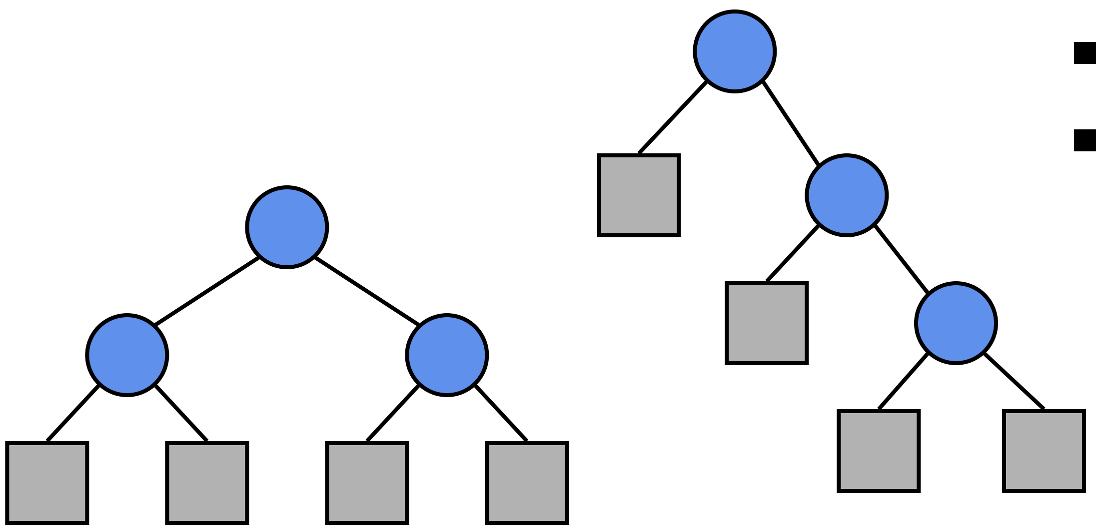
Arbre de decisió

- Es pot associar un arbre binari a un procés de decisió
  - nodes interns: preguntes amb resposta si/no
  - nodes externs: decisions
- Fent un recorregut de l'arbre es pot prendre la decisió
- Exemple: decisió per on anar a sopar



Propietats dels arbres binaris

- **Notació**
  - n nombre de nodes
  - e nombre de nodes externs
  - i nombre de nodes interns
  - h alçada



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

TAD BinaryTree

- El TAD arbre binari (**TAD BinaryTree**) estén el **TAD Tree**
  - hereta tots els mètodes del TAD Tree
- Mètodes addicionals:
  - position p.left()
  - position p.right()
- Arbre binari complet: Cada node té 0, 1 o 2 fills.

Interfície en C++ (no està completa)

```
template <class E>
class Position<E>{
public:
  E& operator*();   // getElement
  Position left() const;
  Position right() const;
  Position parent() const;
  bool isRoot() const;
  bool isExternal() const;
};
```

- Les posicions d'un arbre són els seus nodes
- operator* s'usa per retonar l'element que guarda el node

Interfície en C++ (no està completa)

```
template <class E>
class BinaryTree<E>{
public:
  int size() const;
 bool empty() const;
  Position root() const;
  PositionList positions() const;
};
```

- PositionList segueix l'estàndar TAD llista
  - Es podria implementar amb std::list<Position>

Implementacions del TAD BinaryTree

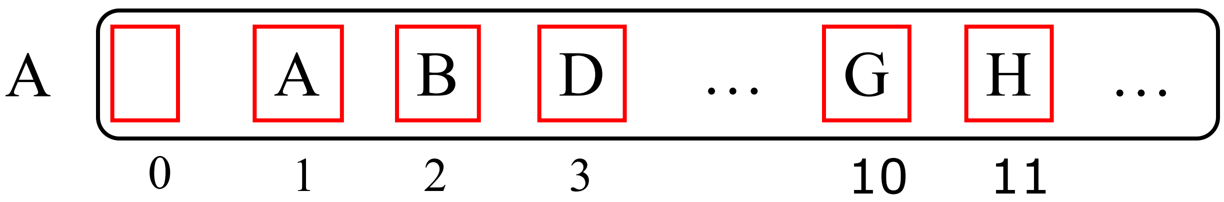
La implementació dels arbres binaris es pot fer de dues maneres:

1. Implementació **usant vectors o arrays**
   - En aquest cas s'ha de definir el màxim nombre d'elements que hi haurà a l'arbre
2. Implementació **usant nodes enllaçats**
   - En aquest cas, no hi ha limitació d'espai

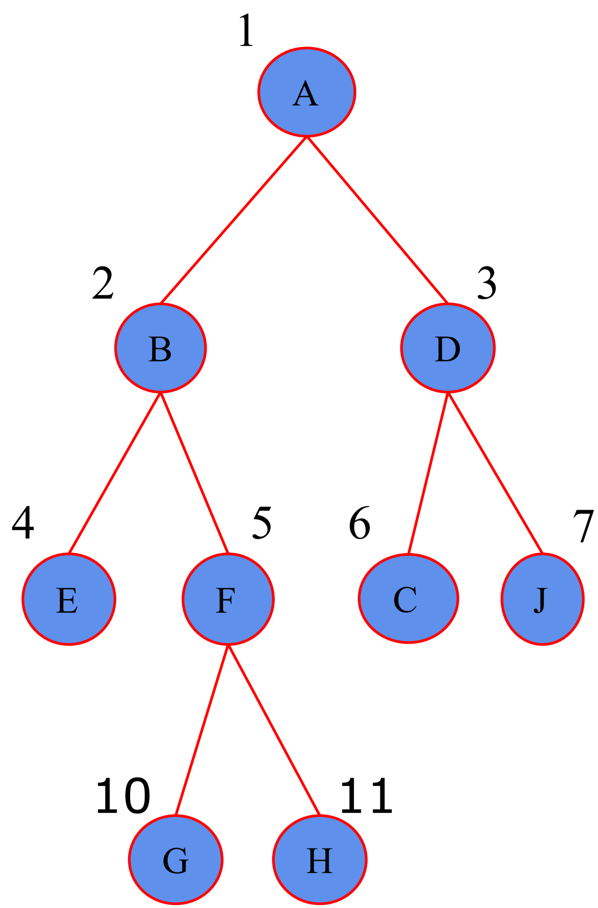
A continuació veurem les dues implementacions

Implementació basada en vectors

- Els nodes es guarden en un array A

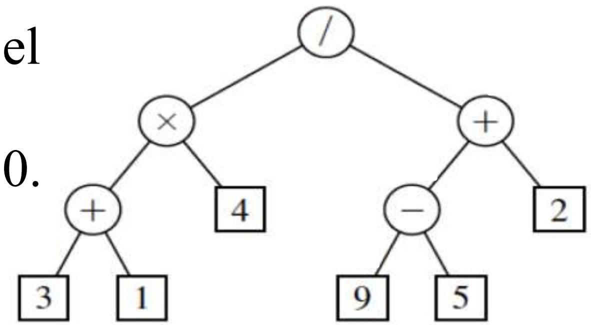


- Els nodes tenen una posició fixa respecte al pare.
  - La posició 0 de l'array A no s'utilitza
  - A[1] guardarà el node arrel (root)
  - si el node és fill esquerra del seu pare
    - si a la casella *i* està el node pare, a la casella A[2*i] estarà el fill esquerra
  - si el node és fill dret del seu pare
    - si a la casella *i* està el node pare, a la casella A[2*i +1] estarà el fill dret



Exercici

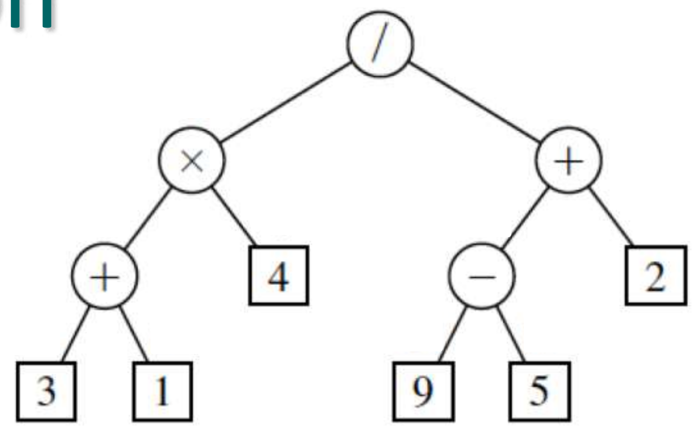
1. Passeu l'arbre del dibuix a les caselles, posant el node arrel a la casella 1.
2. Feu el mateix, posant el node arrel a la casella 0. Definiu la formula d'on trobarà un pare al seu fill esquerra i al seu fill dret.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Solució del segon

- Noteu que hi ha caselles de l'array que queden buides perquè hi ha nodes que no tenen fills

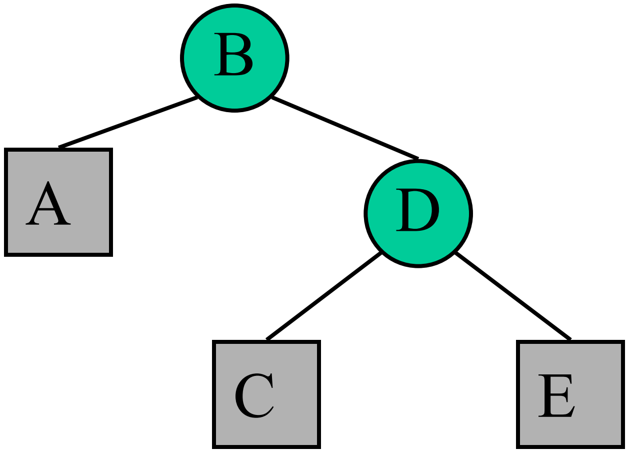


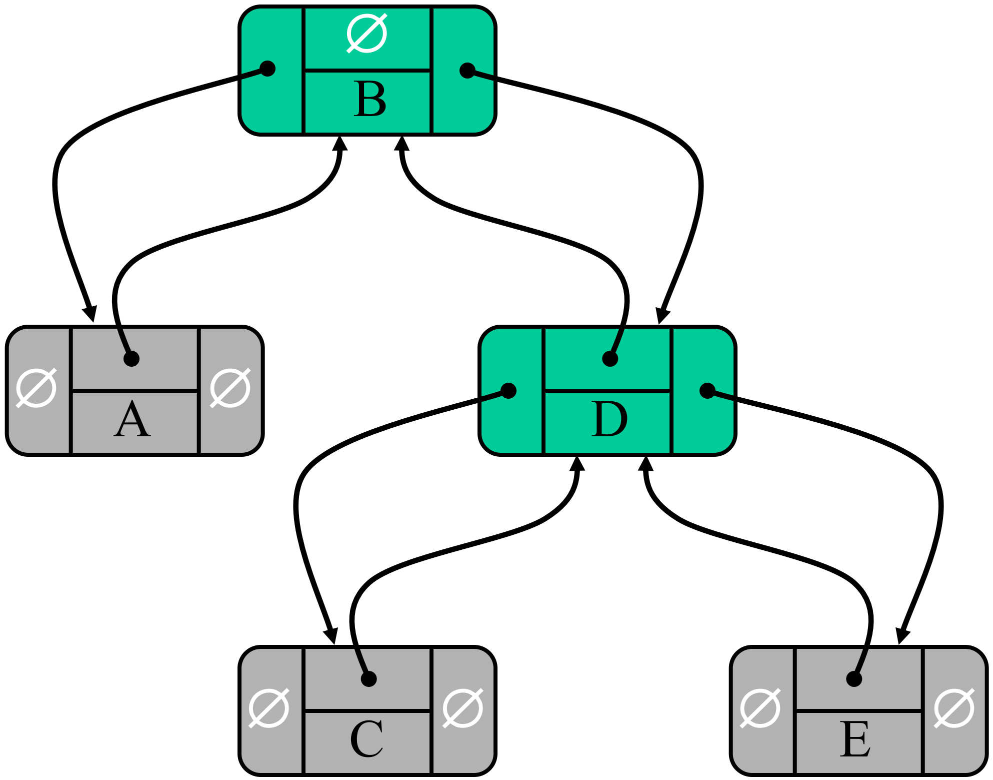
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
/	x	+	+	4	-	2	3	1			9	5		

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

Arbres binaris en estructura encadenada (LinkedBinaryTree)

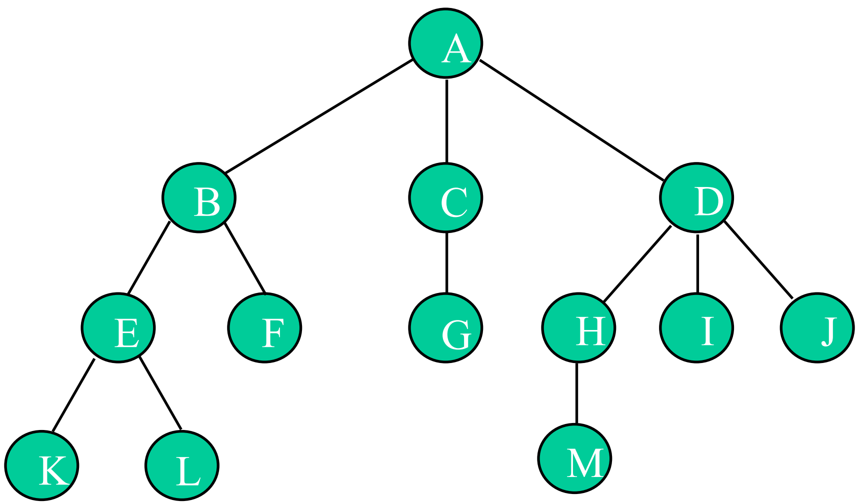
- Un node es representa per un objecte que guarda
  - L'Element
  - El node pare
  - El node esquerra
  - El node dret
- Els objectes Node implementen el TAD Position



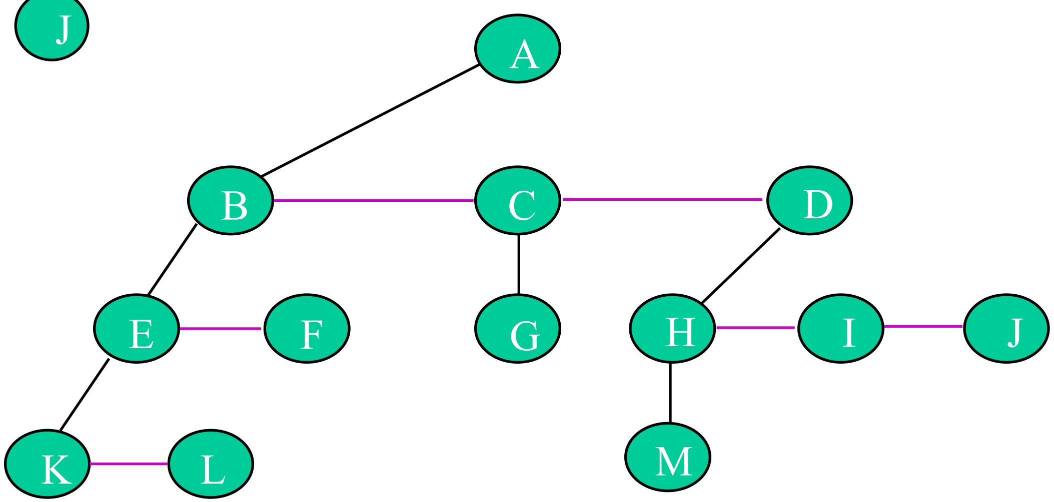


Com representar arbres amb arbres binaris

Propietat: Un arbre de qualsevol grau es pot representar com un arbre binari

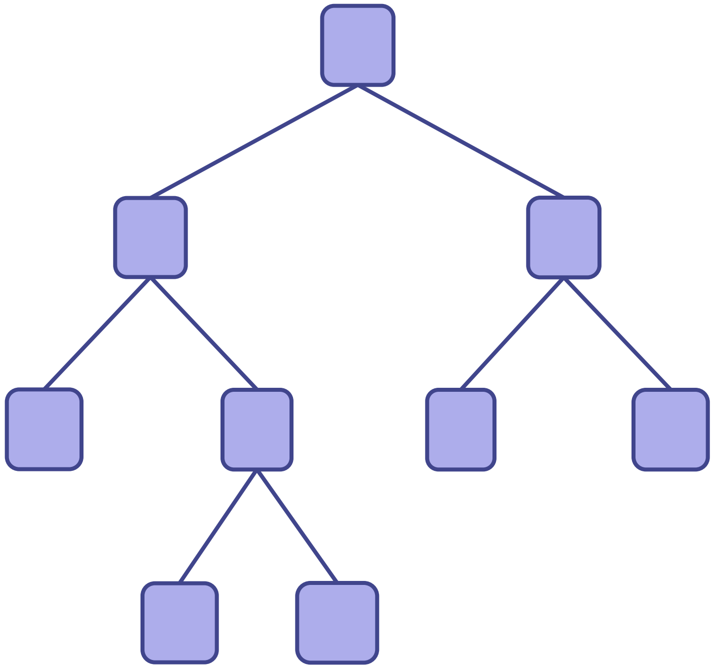


Regla per convertir-lo en binari:
Els fills esquerres continuen sent fills esquerres i els seus germans passen a ser fills drets

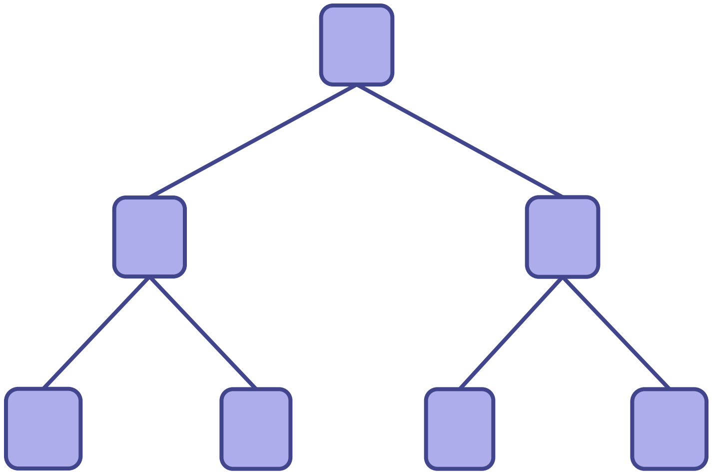


Arbre binari perfecte

- Un arbre binari és **perfecte** si cada nivell està completament ple



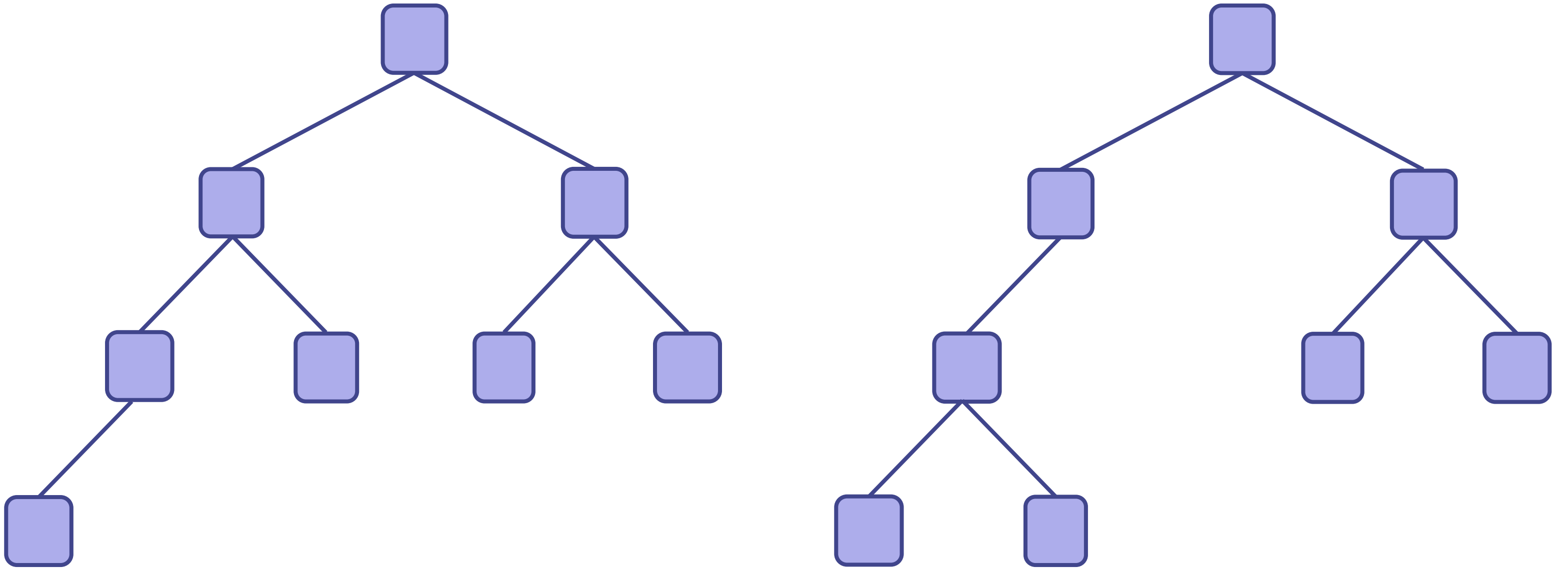
No perfecte



Perfecte!

Complet

- Un arbre binari és **quasi-complet (o complet)** si:
  - Cada nivell està completament ple, excloent el nivell més baix
  - Tots estan el màxim a l'esquerra possible



Quasi-complet! **No complet ni quasi-complet**

UNIVERSITAT DE BARCELONA

Conclusions

- Els arbres binaris utilitzen la mateixa terminologia que els arbres.
- Els arbres complets són arbres que optimitzen l'ús de memòria.
- Hi ha dues maneres per representar un arbre binari:
  - **Representació amb vectors** – òptima per arbres binaris complets.
  - **Representació amb enllaços** – òptima respecte a la inserció i eliminació de nodes i no malgasta memòria.
    - Però necessita gestionar els enllaços dels nodes.

UNIVERSITAT DE BARCELONA

Tema 4 **Estructures No Lineals: Arbres**

Sessió Teo 7

Maria Salamó Llorente

Estructura de Dades

Grau en Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,

Universitat de Barcelona