



# Tema 4 Estructures No Lineals: Arbres

## Sessió Teo 8

**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

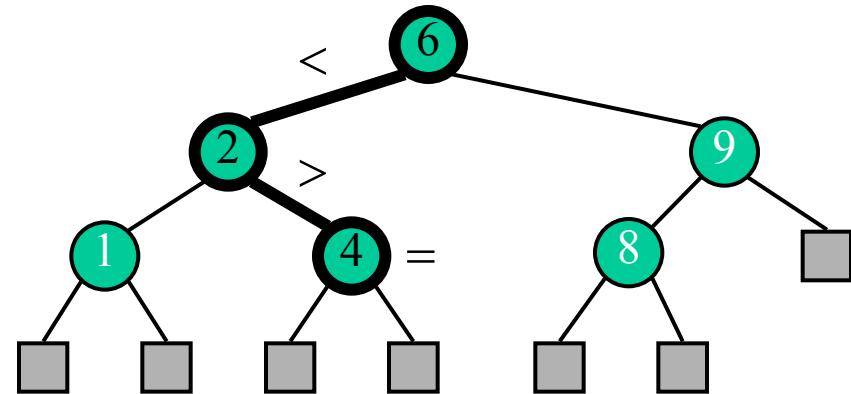
Sessió Teoria 9 (Teo 9)

4.4. Recorreguts en arbres binaris

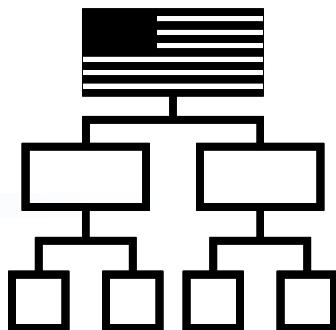
Sessió Teoria 10 (Teo 10)

4.5. Arbres AVL

## 4.3 Arbres binaris de cerca (arbres de cerca binària)



# Arbre binari de cerca

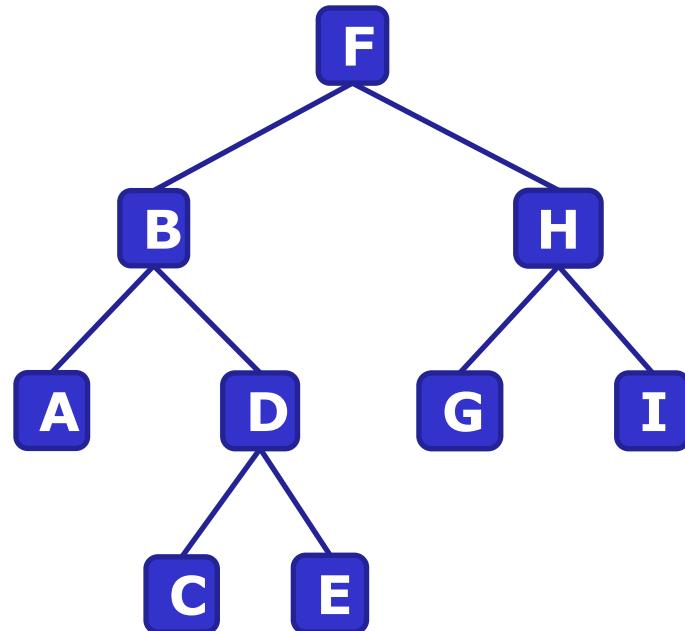


- Un **arbre binari de cerca** (BST) és un arbre binari amb una **propietat d'ordre**:

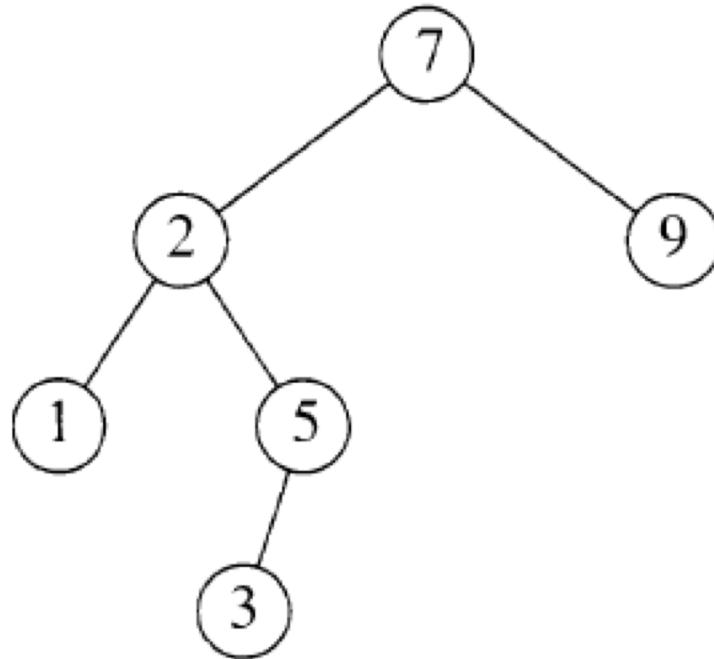
Per cada node:

- Tots els descendents del seu subarbre **esquerra** tenen un valor **menor**
- Tots els descendents del seu subarbre **dret** tenen un valor **major**
- No es permeten elements repetits

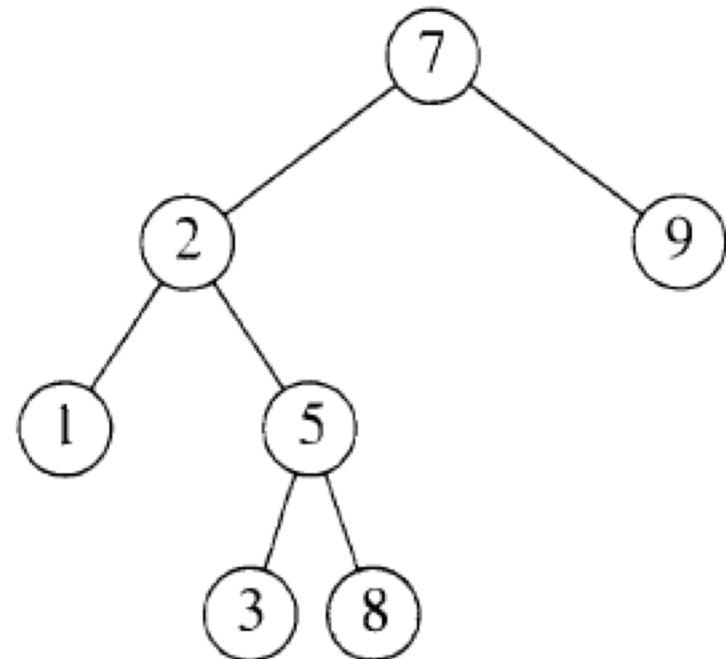
- El **recorregut en inodre** d'un arbre de cerca binària visitarà els valors en ordre creixent



# Arbre binari de cerca



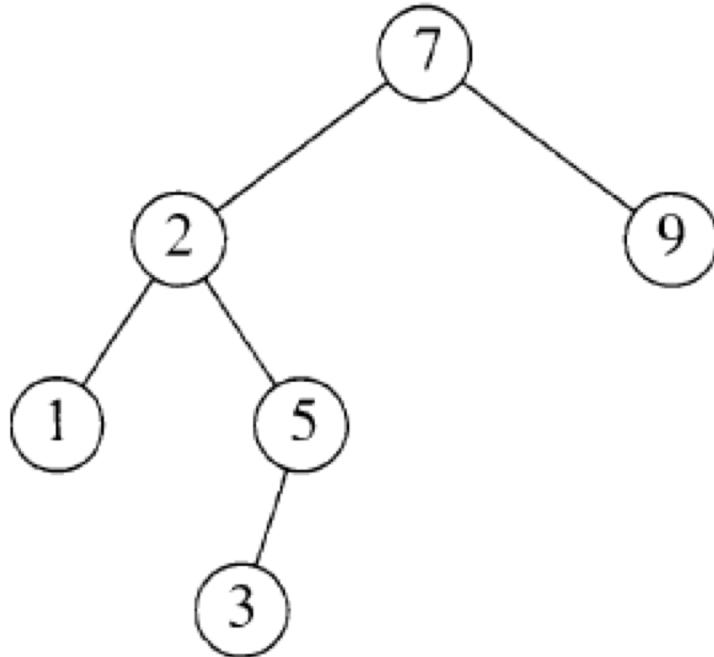
(a)



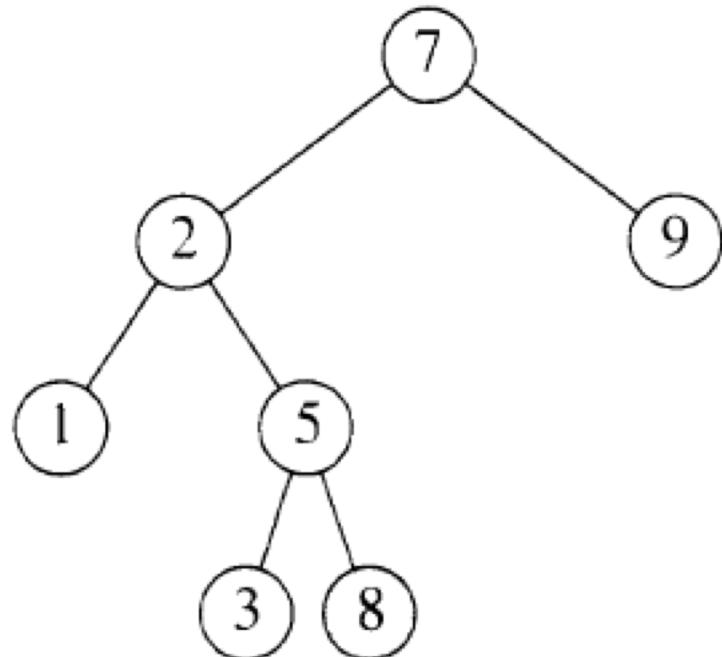
(b)

Quin d'aquests arbres és un arbre binari de cerca?

# Arbre binari de cerca



(a)



(b)

- Dos exemples d'arbres:

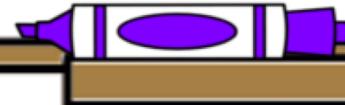
- (a) SI és un arbre binari de cerca (Binary Search Tree - BST)
- (b) NO és un arbre binari de cerca



# Arbre BST

Volem crear un BST on cada node es correspongui amb una paraula de la següent frase:

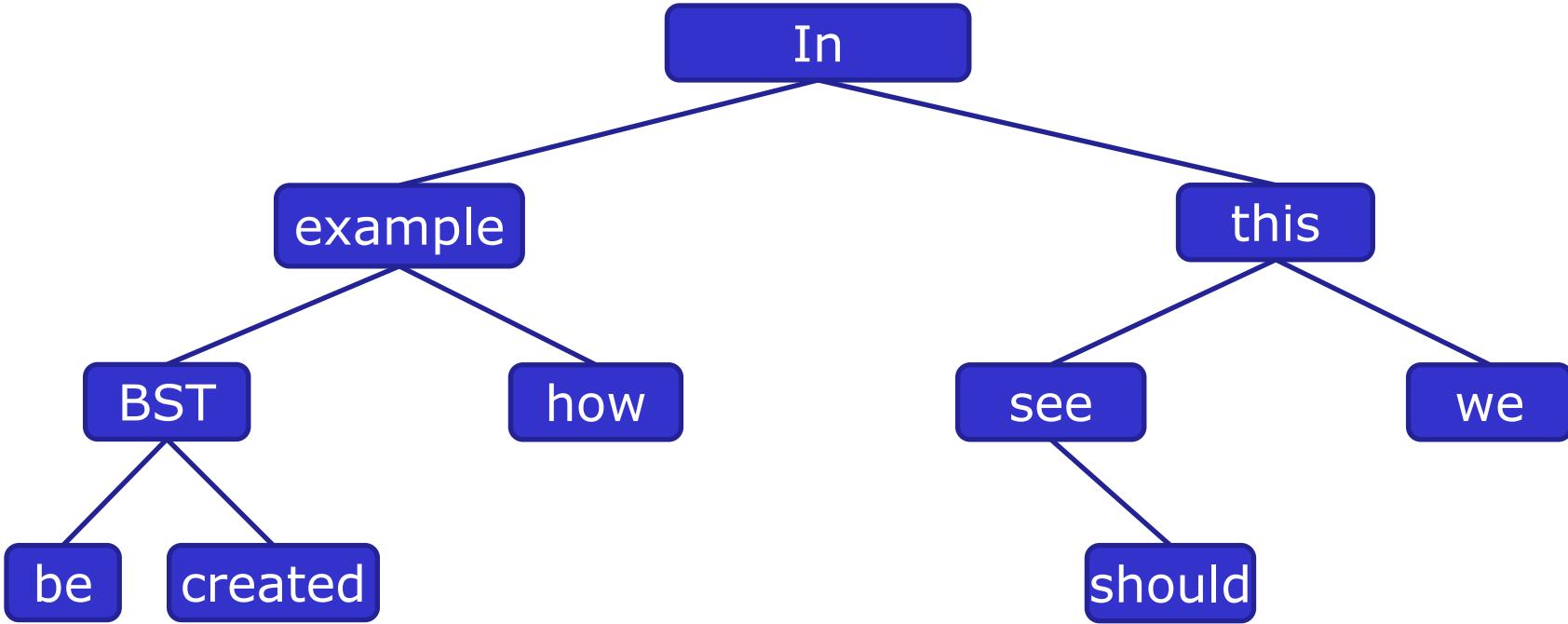
“In this example we see how BST should be created”



# Arbre BST

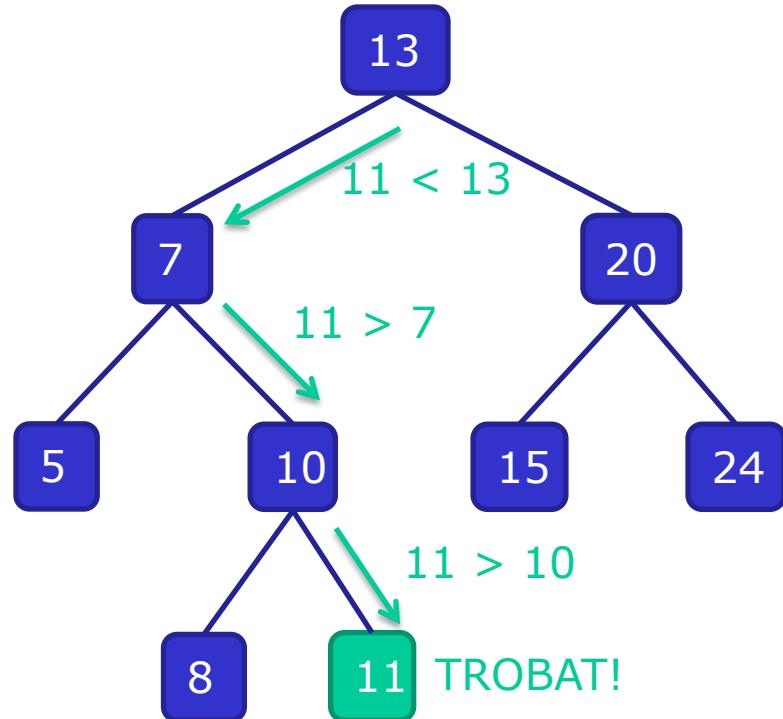
Volem crear un BST on cada node es correspongui amb una paraula de la següent frase:

“In this example we see how BST should be created”



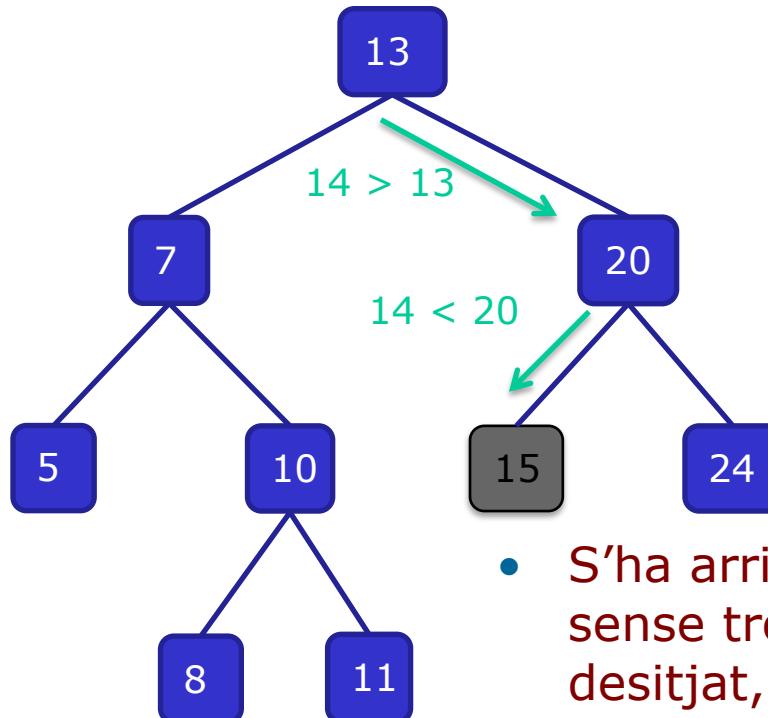
# Cerca en un BST

- Per cercar un valor  $k$ , s'ha de traçar un camí cap a baix començant des de l'arrel
- El següent node a visitar depèn de la comparativa de  $k$  amb el valor del node actual
- **Exemple:** volem trobar el valor 11 en l'arbre



# Cerca en un BST

- **Que ocorre si l'element no es troba en l'arbre?**
- Imagina que s'està cercant el número 14

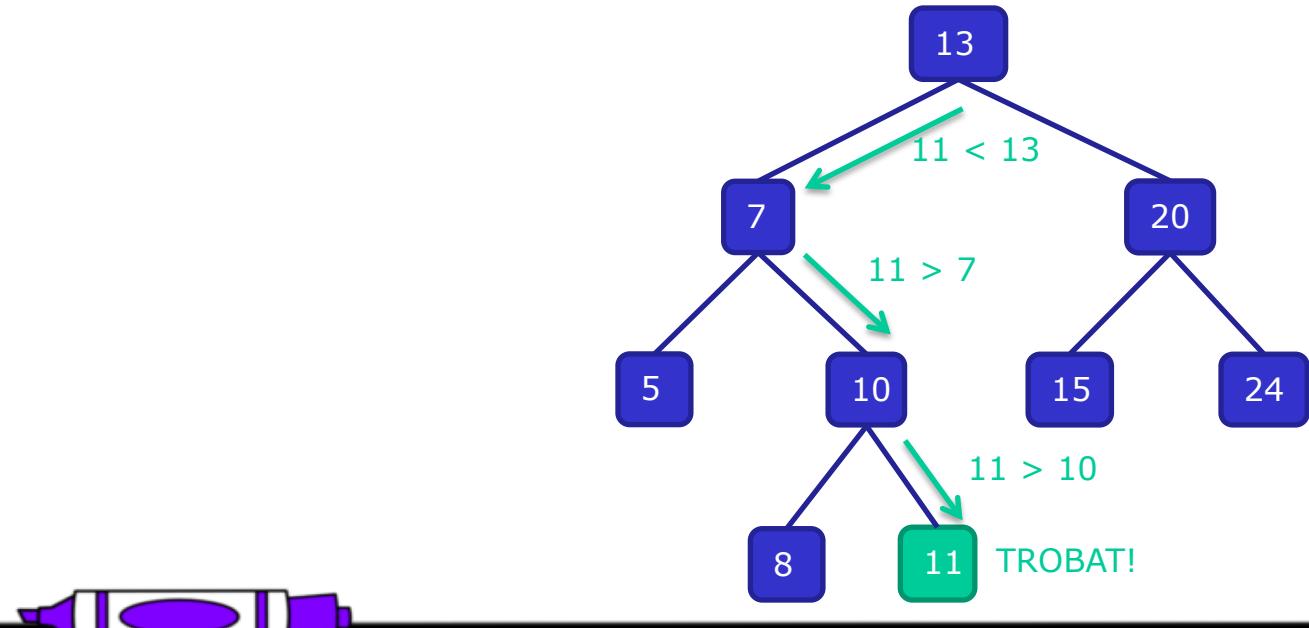


- S'ha arribat a una fulla sense trobar el número desitjat, per tant el número no es troba en l'arbre

# Cerca en BST: Pseudocodi

- Fer com exercici amb una implementació **RECURSIVA**

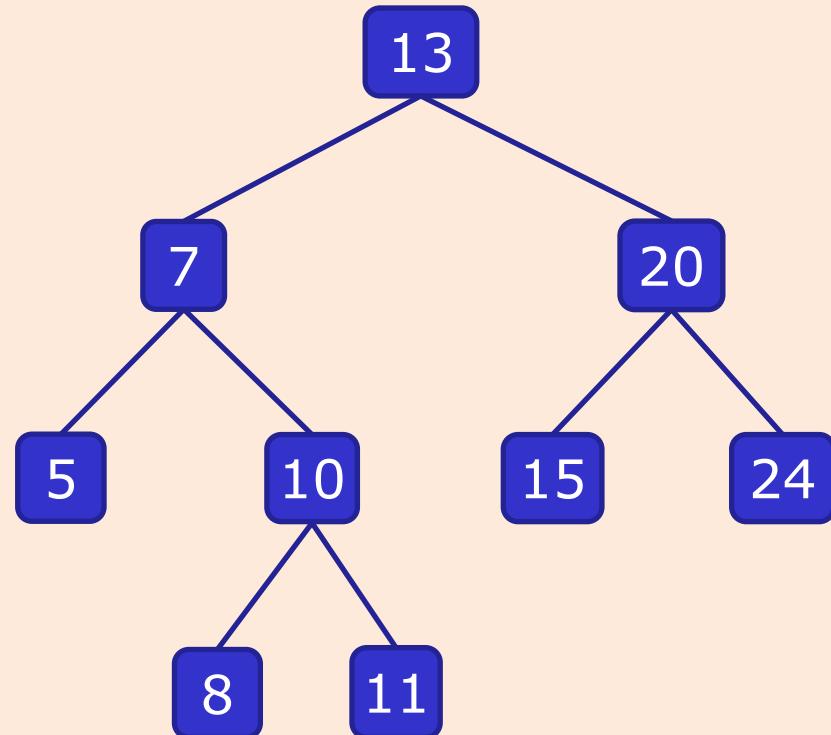
**Algorithm TreeSearch( Key  $k$ , Node  $v$ )** // cerca element  $k$  a partir del node  $v$



# Afegir en un BST

- Per afegir en un BST s'utilitza la mateixa estratègia usada en la cerca
- **Un nou ítem sempre s'afegeix com una nova fulla**

Arbre inicial:  
-> Afegir **element 17**

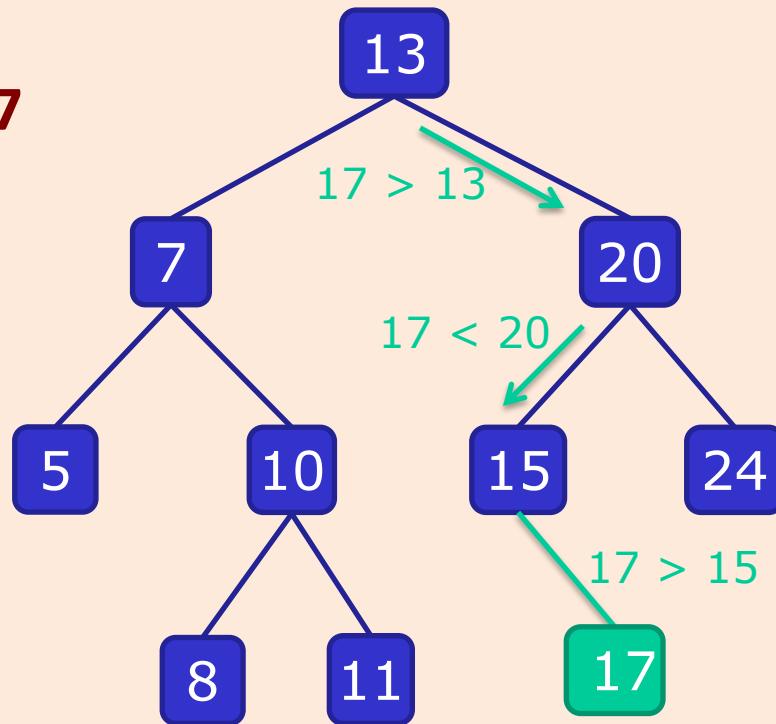


# Afegir en un BST

- Per afegir en un BST s'utilitza la mateixa estratègia usada en la cerca
- **Un nou ítem sempre s'afegeix com una nova fulla**

Arbre final:

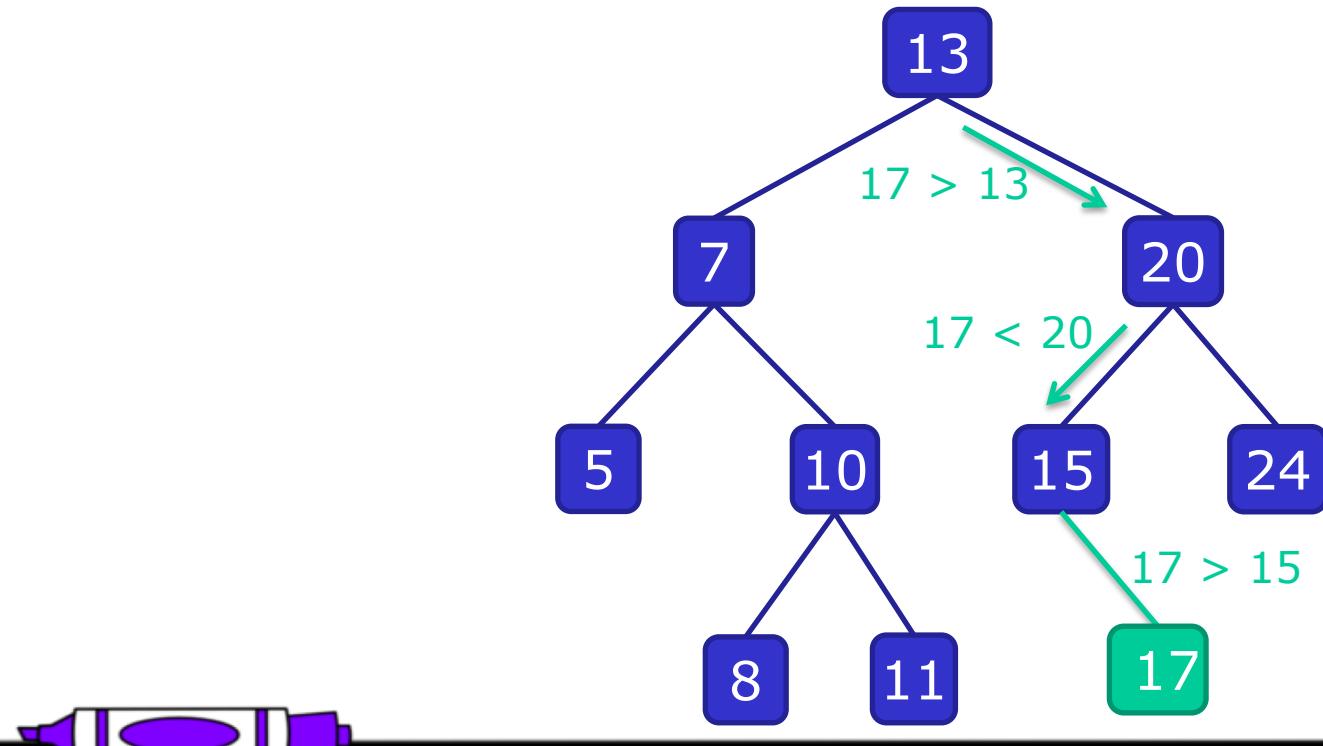
- després d'afegir l' element 17



# Afegeix en BST: Pseudocodi

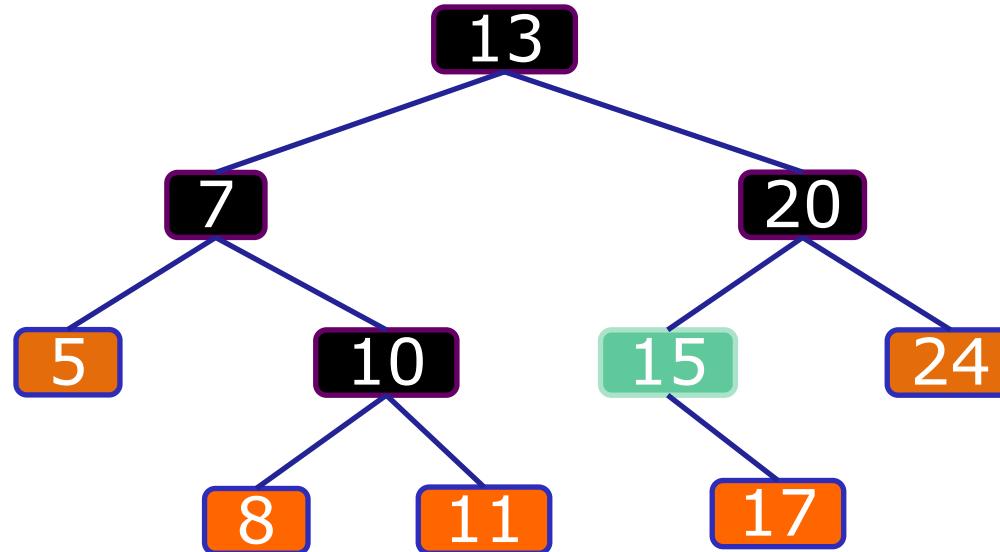
- Fer com exercici amb una implementació **RECURSIVA**

Algorithm insert(Position node, Element ele)



# Eliminar en un BST

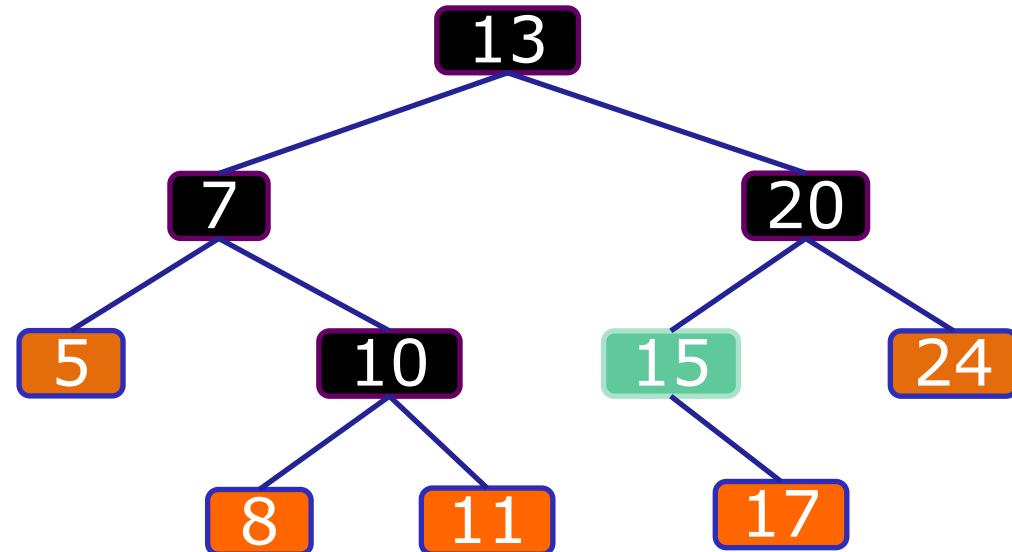
- Eliminar elements en un ABB és un algorisme més complexe
- S'han de considerar **3 casos**:
  - **CAS 1.** Eliminar una **fulla**. S'elimina el node corresponent
  - **CAS 2.** Eliminar un **node intern amb un fill**
  - **CAS 3.** Eliminar un **node intern amb dos fills**



# Eliminar en un BST: Cas 1

## Cas 1: Eliminar un node fulla

- Estratègia General:
  - S'elimina l'element corresponent i el node pare posa el seu fill a nullptr
- Exemple: **remove(5)**

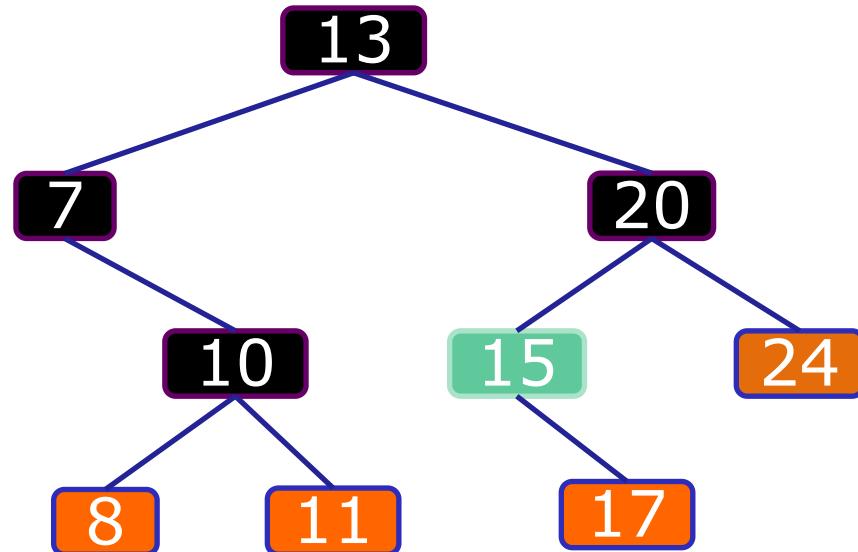


# Eliminar en un BST: Cas 1

## Cas 1: Eliminar un node fulla

- Estratègia General:
  - S'elimina l'element corresponent i el node pare posa el seu fill a nullptr
- Exemple: **remove(5)**

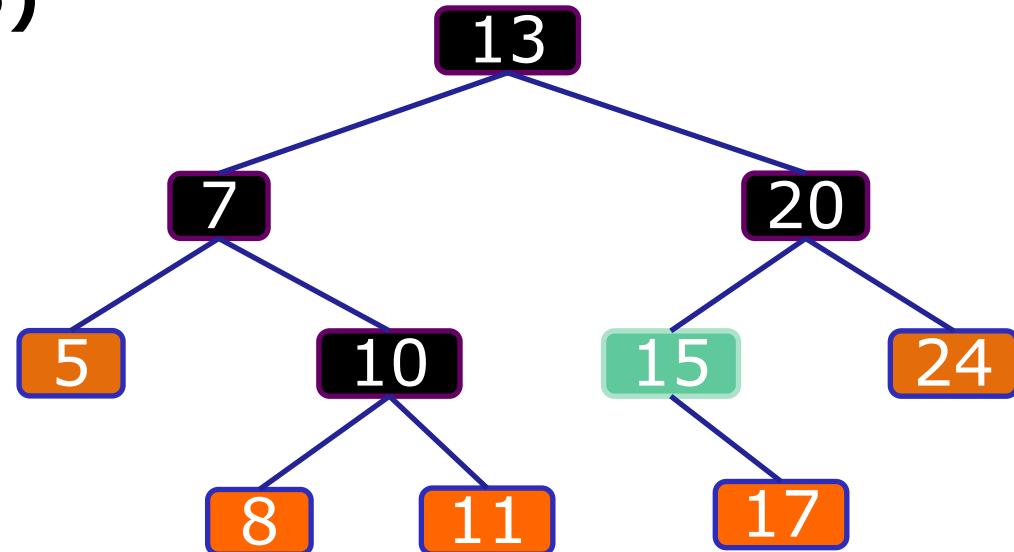
1. S'elimina el node 5
2. S'assigna com a fill esquerra del node (7) un *nullptr*



# Eliminar en un BST: Cas 2

## Cas 2: Eliminar un node intern amb un fill

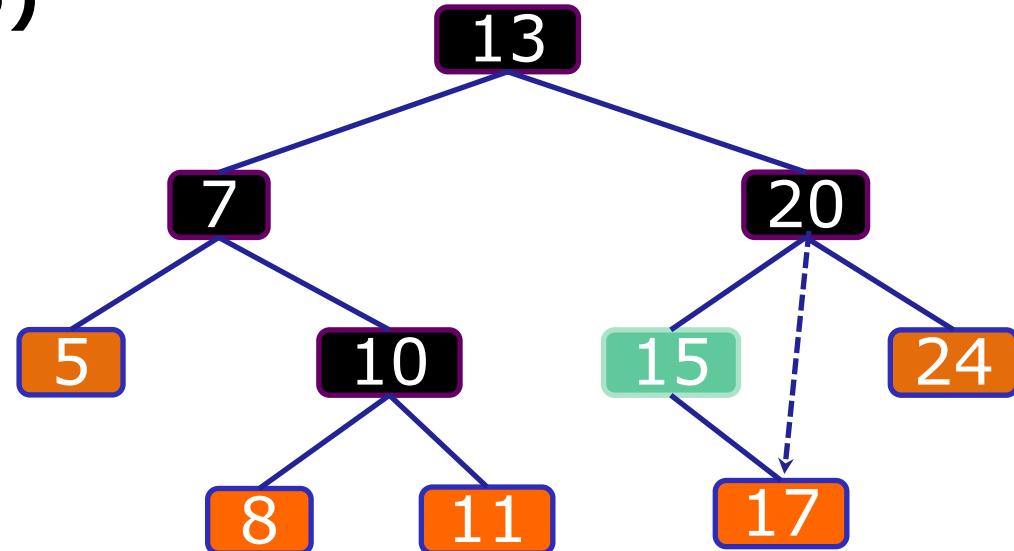
- Estratègia General:
  - S'elimina l'element corresponent connectant el node del pare amb el node del fill del node a eliminar
- Exemple: **remove(15)**



# Eliminar en un BST: Cas 2

## Cas 2: Eliminar un node intern amb un fill

- Estratègia General:
  - S'elimina l'element corresponent connectant el node del pare amb el node del fill del node a eliminar
- Exemple: **remove(15)**
  1. S'assigna com a fill esquerra del node (20) pare la referència del fill del node 15



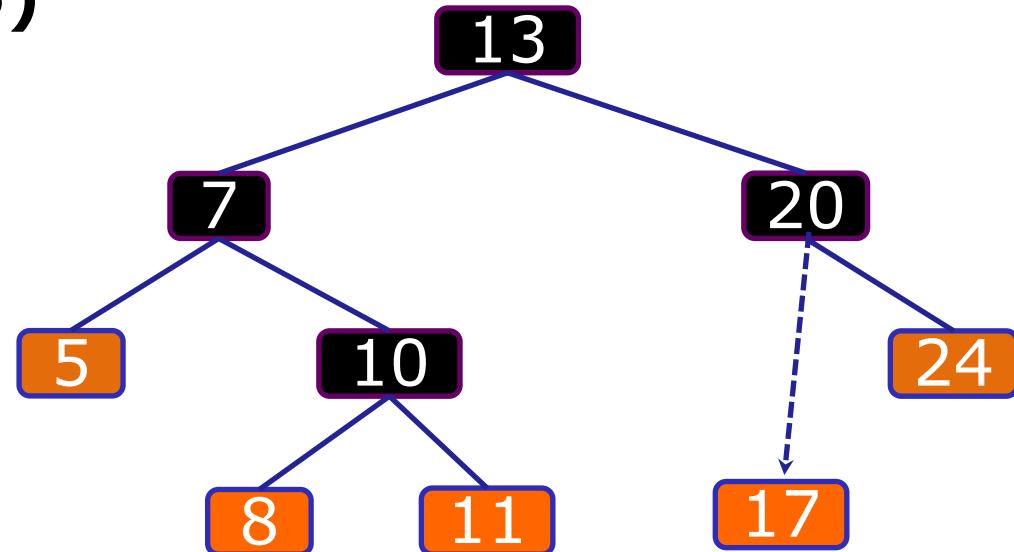
# Eliminar en un BST: Cas 2

## Cas 2: Eliminar un node intern amb un fill

- Estratègia General:
  - S'elimina l'element corresponent connectant el node del pare amb el node del fill del node a eliminar

- Exemple: **remove(15)**

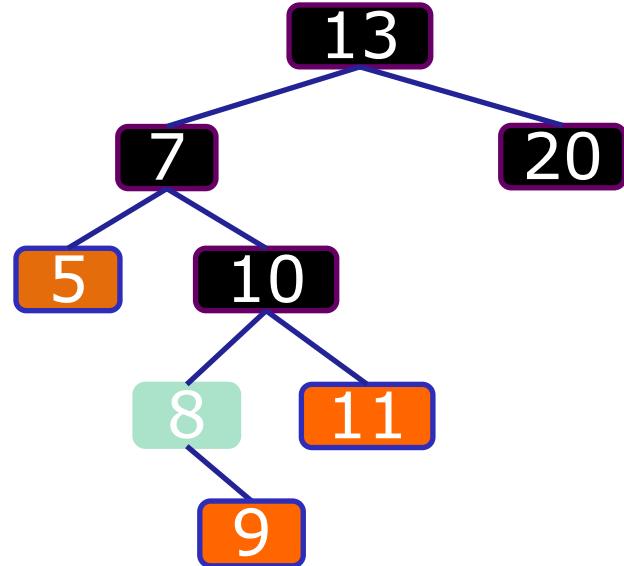
1. S'assigna com a fill esquerra del node (20) pare la referència del fill del node 15
2. S'elimina el node 15



# Eliminar en un BST: Cas 3

## Cas 3: Eliminar un node intern amb dos fills

- Estratègia General:
  - Canviar les dades del node a eliminar per les dades del successor del node
    - **Es canvia pel node amb valor menor de tot el subarbre dret**
  - Eliminar el node successor



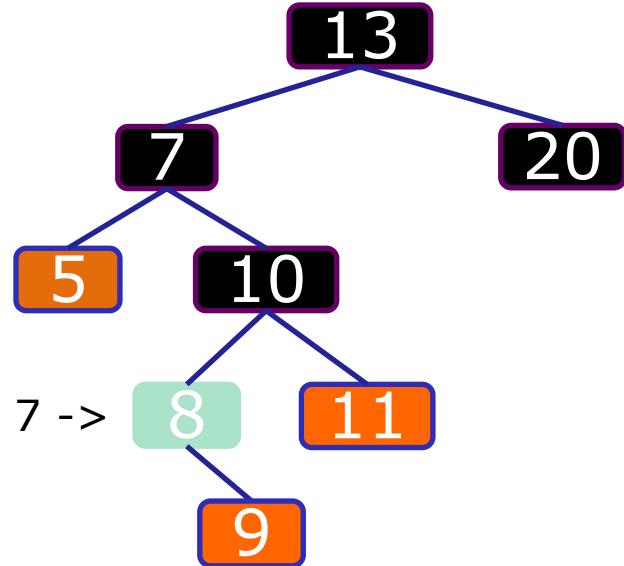
# Eliminar en un BST: Cas 3

## Cas 3: Eliminar un node intern amb dos fills

- Exemple: **remove(7)**
  - Primer busquem el successor del node
    - El successor del node 7 és el node 8

```
void successor (Position node):  
    // Input: node - el node del  
    // que estem buscant el successor  
    Position curr = node.right  
    while (curr.left != null):  
        curr = curr.left  
    return curr
```

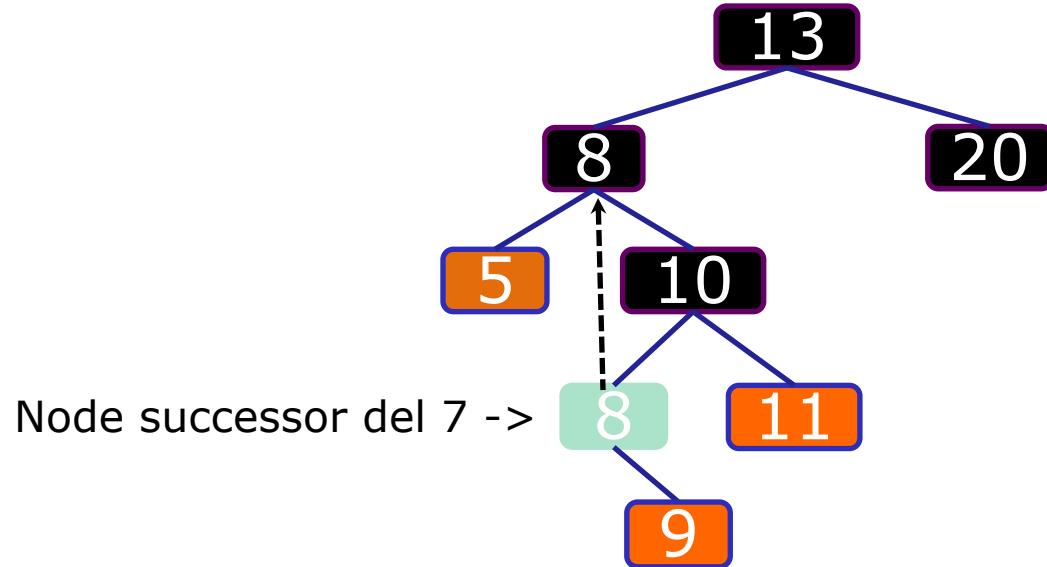
Node successor del 7 ->



# Eliminar en un BST: Cas 3

## Cas 3: Eliminar un node intern amb dos fills

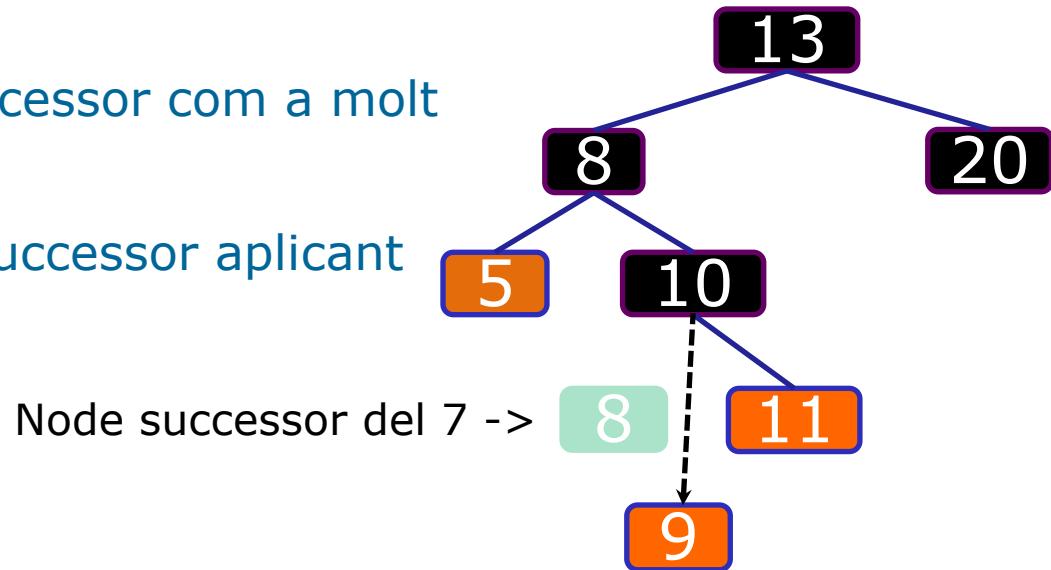
- Exemple: remove(7)
  - Segon, es reemplaça les dades del node a eliminar per les dades del successor



# Eliminar en un BST: Cas 3

## Cas 3: Eliminar un node intern amb dos fills

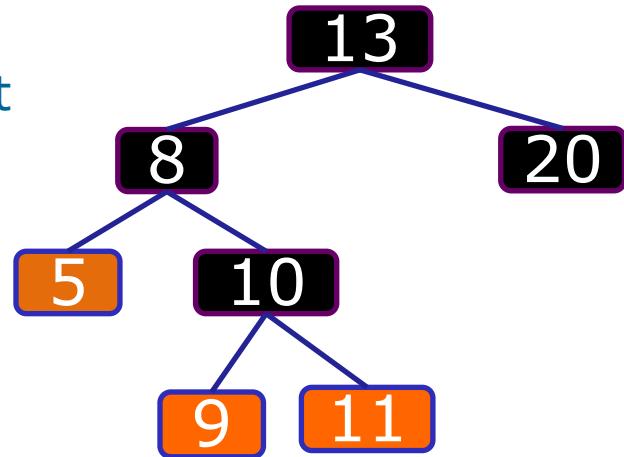
- Exemple: **remove(7)**
  - Per últim, s'ha d'eliminar el node successor
- **IMPORTANT!!** Tenim la seguretat que el node successor NO té fill esquerra
  - Per tant el node successor com a molt té un fill dret
  - Podem eliminar el successor aplicant el cas 1 o el cas 2



# Eliminar en un BST: Cas 3

## Cas 3: Eliminar un node intern amb dos fills

- Exemple: **remove(7)**
  - Per últim, s'ha d'eliminar el node successor
- **IMPORTANT!!** Tenim la seguretat que el node successor NO té fill esquerra
  - Per tant el node successor com a molt té un fill dret
  - Podem eliminar el successor aplicant el cas 1 o el cas 2





# Eliminar en BST: Pseudocodi

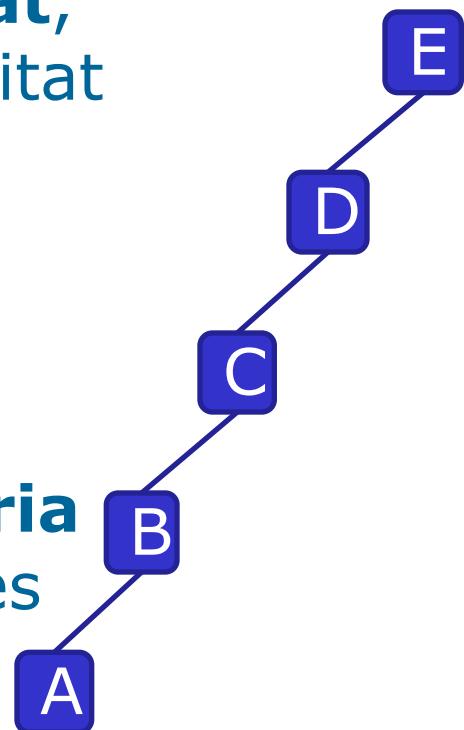
- Fer com exercici amb una implementació **RECURSIVA**

Algorithm remove(Position node)



# Rendiment en un BST

- Depèn de l'alçada de l'arbre. En el **pitjor cas**, tindrem un arbre amb **l'alçada igual que el nombre de nodes**
- Si l'arbre està **perfectament balancejat**, l'alçada és  $\log_2 n$ , el que fa que la coplexitat de les funcions sigui  **$O(\log_2 n)$ . MENYS que lineal!!**
- En el cas d'un **arbre totalment desbalancejat, l'arbre de cerca binària és una Ordered Linked List**, i les seves funcions tenen una complexitat  $O(n)$





# Exercici

- Dibuixa l'arbre de cerca binària que resulta d'afegir els següents elements:  
**36, 72, 44, 12, 1, 25, 50, 85, 77, 60**  
en un arbre de cerca binària inicialment buit
- Contesta a les següents preguntes:
  1. Quina alçada té l'arbre ?
  2. Quina és la profunditat del node 50?
  3. Quina és l'aritat de l'arbre?





# Tema 4 Estructures No Lineals: Arbres

## Sessió Teo 8

**Maria Salamó Llorente**  
**Estructura de Dades**

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