Arbres

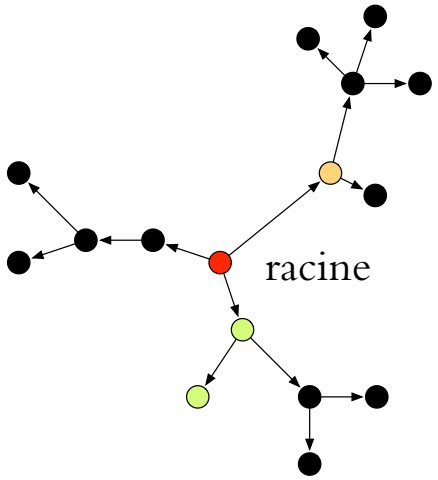
Arbres — terminologie

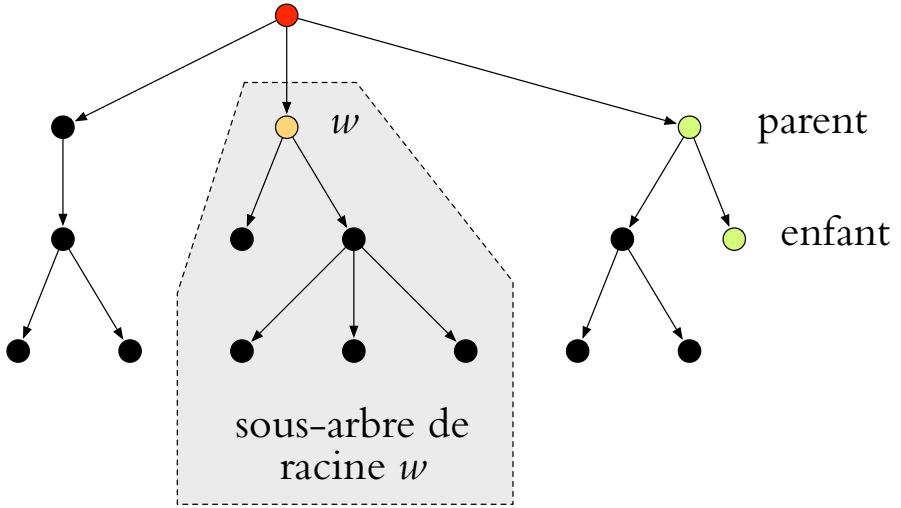
Graphe : sommet et arêtes

Arbre (en théorie des graphes) : graphe non-orienté, connexe et acyclique

Arbre raciné ou **arborescence** (en théorie des graphes) : graphe orienté, et connexe avec un sommet spécial, la **racine**, dans lequel il y a 1 chemin simple de la racine à chaque sommet.

- orientation des arêtes : relations parent-enfant
- u est dans le sous-arbre enraciné à w ssi w est sur le chemin de la racine à u





Arbres — terminologie

Ancêtre : w est l'ancêtre de u ssi u est dans le sous-arbre de racine w

Descendant : u est un descendant de w ssi u est dans le sous-arbre de racine w

Dans des graphes orientés (y inclut les arborescences) les sommets s'appellent aussi des nœuds, et les arêtes s'appellent aussi des arcs

Degré d'un nœud : nombre d'arcs sortants (ou nombre d'enfants)

Nœud externe ou feuille : aucun arc sortant (aucun enfant)

Nœud interne : tous les autres (au moins 1 enfant)

Hauteur et profondeur

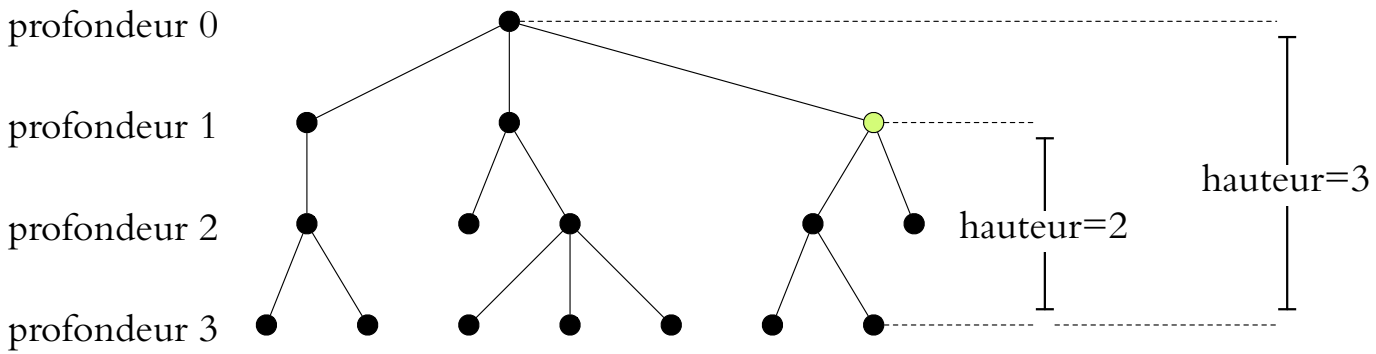
Profondeur d'un nœud u : longueur du chemin de la racine à u

Hauteur d'un nœud u : longueur du chemin à la feuille la plus distante dans le sous-arbre de u

Hauteur de l'arbre : hauteur de la racine

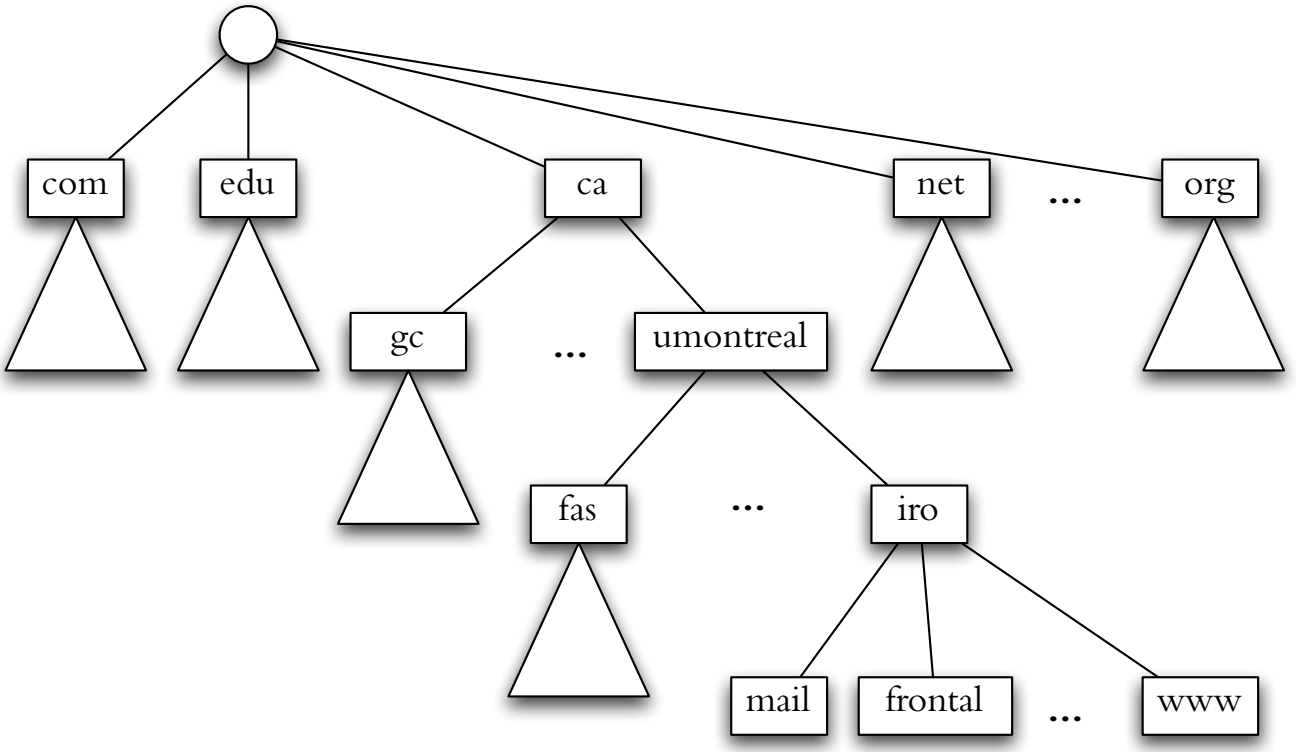
Profondeur de l'arbre : profondeur maximale

Thm. profondeur de l'arbre = hauteur de l'arbre.



Arborescence — exemples

Domaines internet :



Packages en Java, répertoires sous Unix, . . .

Arbre numeroté

Arbre numeroté : les enfants d'un nœud sont étiquetés par des entiers positifs distincts.

i-ème enfant absent : si aucun enfant n'est étiqueté par i

Arité k : ssi tous les enfants avec étiquettes $> k$ sont absents.

Arbre binaire : arbre numeroté d'arité 2

enfant gauche ou droit : enfant étiqueté par 1 ou 2

frères ou sœurs : nœuds avec le même parent

Arbre numeroté (cont.)

Définition altérnative par récurrence :

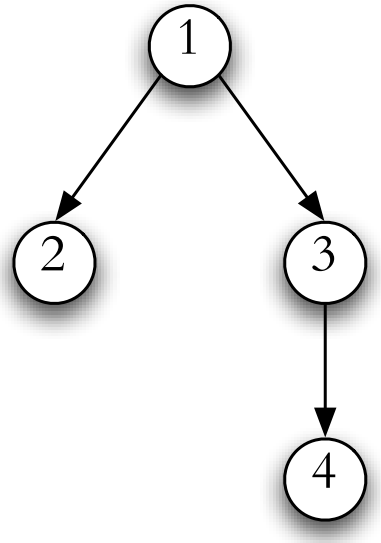
Déf. *Un arbre k-aire T est une structure définie sur un ensemble fini de nœuds qui*

1. *ne contient aucun nœud,* ***ou***
2. *est composé de $(k+1)$ ensembles de nœuds disjoints : un nœud* ***racine*** *r, et les arbres k-aires $T_1, T_2, \ldots, T_k$.*

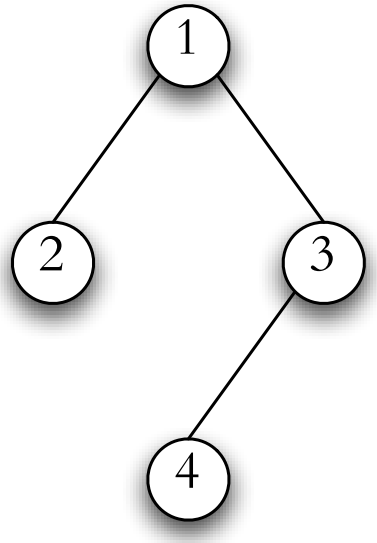
(En 2, la racine de T_i quand T_i est non-vide est l'enfant de r étiqueté par i.)

Arbre numeroté (cont.)

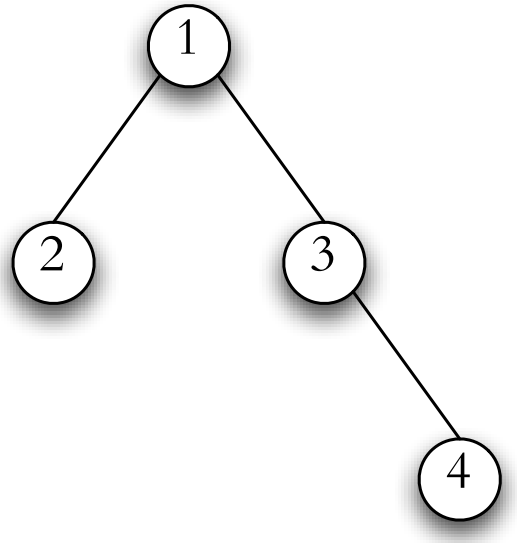
Attention : l'ordre des enfants est important dans un arbre numeroté



arborescence
(l'ordre des enfants
n'est pas important)



arbre binaire
(«4» est l'enfant gauche)



arbre binaire
(«4» est l'enfant droit)

Implantation d'un arbre numeroté

Arbre = ensemble d'objets représentant de nœuds + relations parent-enfant

En général, on veut retrouver facilement le parent et les enfants de n'importe quel nœud

Approche Java :

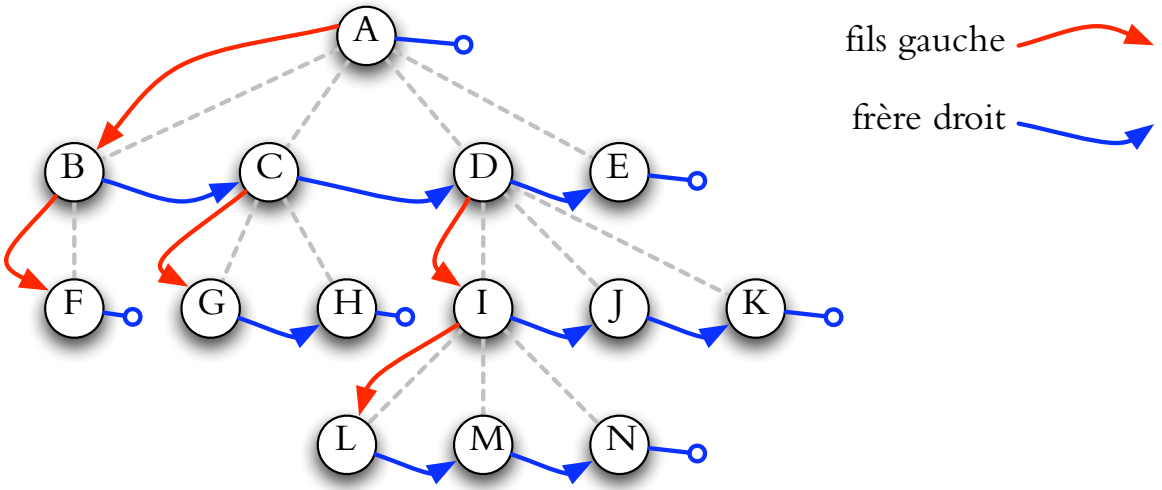
```
public class TreeNode {
   TreeNode parent;
   TreeNode enfant_gauche;
   TreeNode enfant_droit;
   ...
}
```

Si arbre k-aire, alors on peut avoir `TreeNode[] enfants`
avec `enfants.length` $= k$.

Implantation (cont)

Si l'arité de l'arbre n'est pas connu en avance (ou la plupart des nœuds ont très peu d'enfants), on peut utiliser

1. une liste pour stocker les enfants
2. représentation **fils-gauche**, **frère-droit** (*first-child, next-sibling*)



Parcours des arbres

Dans un parcours, tous les nœuds de l'arbre sont visités.

Déf. Dans un parcours préfixe (*preorder traversal*), chaque nœud est visité avant que ses enfants soient visités.

Déf. Dans un parcours postfixe (*postorder traversal*), chaque nœud est visité après que ses enfants sont visités.

Parcours préfixe et postfixe

```
   Algo PARCOURS-PRÉFIXE(x)
1  if x ≠ null then
2      imprimer de x
3      for i ← 1, ..., k do PARCOURS-PRÉFIXE(enfant(x, i))
```

```
   Algo PARCOURS-POSTFIXE(x)
1  if x ≠ null then
2      for i ← 1, ..., k do PARCOURS-POSTFIXE(enfant(x, i))
3      imprimer de x
```

(enfant(x, i) donne l'enfant de x étiqueté par i — s'il n'y en a pas, alors null)

Maintenant PARCOURS-PRÉFIXE(racine) va imprimer tous les nœuds dans l'arbre dans l'ordre préfixe.

Parcours infixe

On peut parcourir un arbre binaire aussi dans l'ordre infixe

Déf. Dans un parcours infixe (*inorder traversal*), chaque nœud est visité après son enfant gauche mais avant son enfant droit.

```
   Algo PARCOURS-INFIXE(x)
1  if x ≠ null then
2      PARCOURS-INFIXE(gauche(x))
3      imprimer x
4      PARCOURS-INFIXE(droit(x))
```

Arbre binaire — notation

Dans nos arbres binaires, chaque nœud a une **valeur** (ou clé).

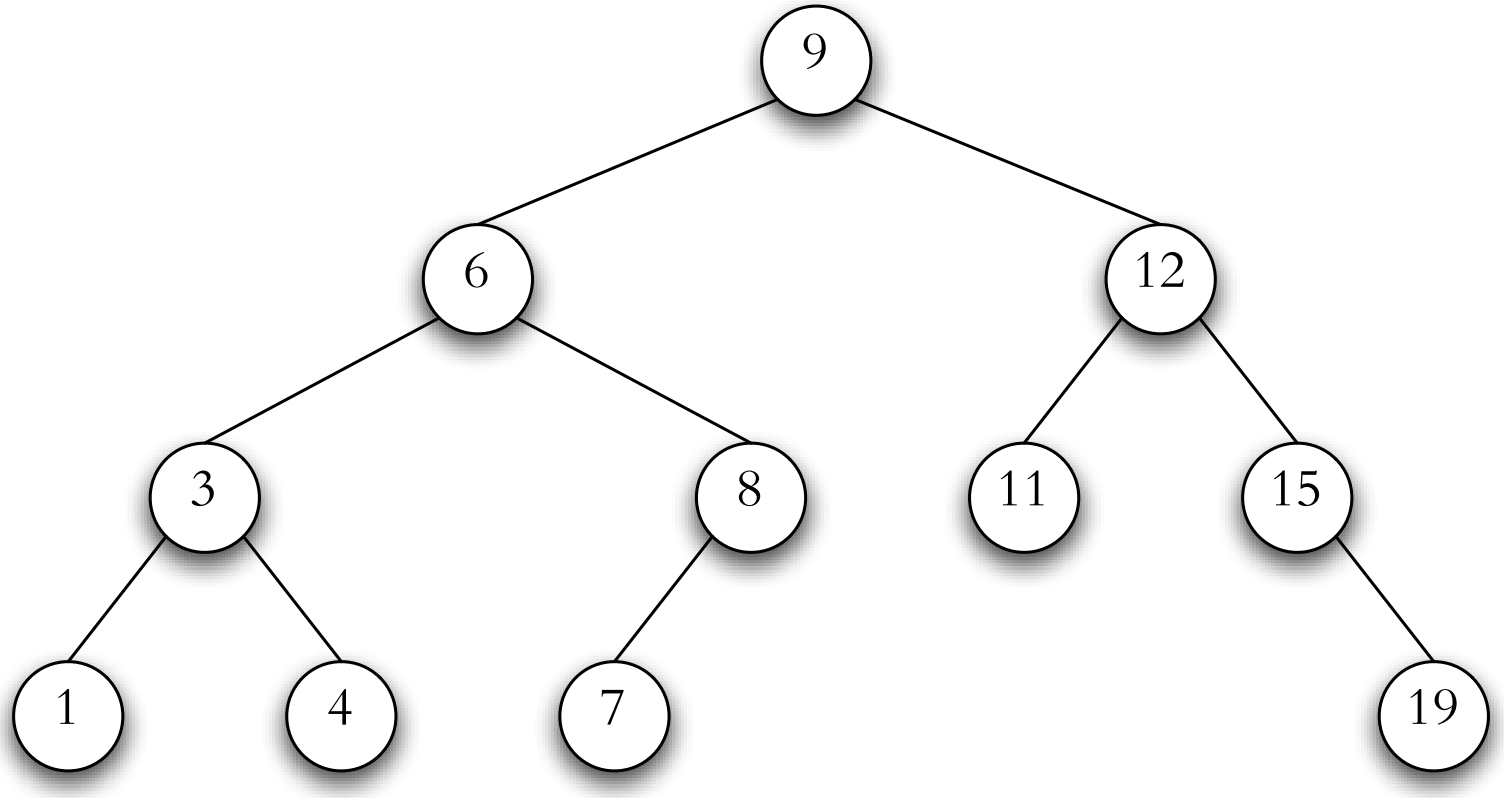
Objets pour notre ADT : nœuds (ensemble $\mathcal{N}$) et arbres binaires (ensemble $\mathcal{T}$)

Opérations sur nœuds :
$\mathsf{gauche}(x)$ et $\mathsf{droit}(x)$ pour les enfants de x (null s'il n'y en a pas)
$\mathsf{parent}(x)$ pour le parent de x (null pour la racine)
$\mathsf{val}(x)$ pour la valeur de nœud x (en général, un entier dans nos discussions)

Déf. Un arbre binaire est un arbre de recherche ssi les nœuds sont énumerés lors d'un parcours infixe en ordre croissant de valeur.

Thm. Soit x un nœud dans un arbre binaire de recherche. Si y est un nœud dans le sous-arbre gauche de x, alors $\mathsf{val}(y) \leq \mathsf{val}(x)$. Si y est un nœud dans le sous-arbre droit de x, alors $\mathsf{val}(y) \geq \mathsf{val}(x)$.

Arbre de recherche — exemple



Arbre de recherche (cont)

À l'aide d'un arbre de recherche, on peut implanter des opérations sur les ensembles (éléments = valeurs des nœuds) d'une manière très efficace.

Opérations : **recherche** d'une valeur particulière, **insertion** ou **suppression** d'une valeur, recherche de **min** ou **max**, et des autres.

Min et max

```
    Algo MIN() // trouve la valeur minimale dans l'arbre
1   x ← racine; y ← null
2   while x ≠ null do
3       y ← x; x ← gauche(x)
4   retourner y
```

```
    Algo MAX() // trouve la valeur maximale dans l'arbre
1   x ← racine
2   while x ≠ null do
3       y ← x; x ← droit(x)
4   retourner y
```

Recherche

```
     Algo FIND(x, v) // trouve la valeur v dans le sous-arbre de x
F1   if x = null ou v = val(x) alors retourner x
F2   if v < val(x)
F3   then retourner FIND(gauche(x), v)
F4   else retourner FIND(droit(x), v)
```

Maintenant, FIND(racine, v) retourne
- soit un nœud dont la valeur est égale à v,
- soit null.

Notez que c'est une recursion terminale $\Rightarrow$ transformation en forme itérative

Recherche (cont)

Solution itérative (plus rapide) :

```
     Algo FIND(x, v) // trouve la valeur v dans le sous-arbre de x
F1   while x ≠ null et v ≠ val(x) do
F2       if v < val(x)
F3       then x ← gauche(x)
F4       else x ← droit(x)
F5   retourner x
```

Recherche — efficacité

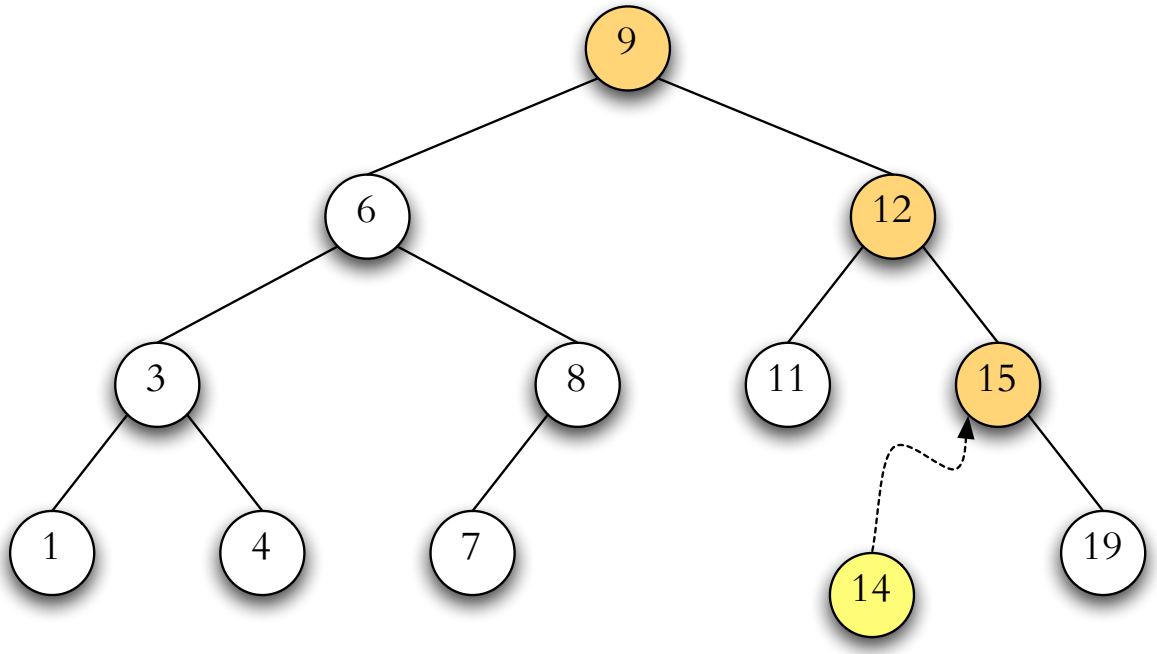
Dans un arbre binaire de recherche de hauteur h :

MIN() prend $O(h)$
MAX() prend $O(h)$
FIND(racine, v) prend $O(h)$

Insertion

On veut insérer une valeur v

Idée : comme en FIND, on trouve la place pour v (enfant gauche ou droit manquant)



insertion de «14»

Insertion (cont.)

insertion — pas de valeurs dupliquées

```
     Algo INSERT(v) // insère la valeur v dans l'arbre
 I1  x ← racine
 I2  if x = null then initialiser avec une racine de valeur v et retourner
 I3  while vrai do // (conditions d'arrête testées dans le corps)
 I4      if v = val(x) then retourner // (pas de valeurs dupliquées)
 I5      if v < val(x)
 I6      then if gauche(x) = null
 I7          then attacher nouvel enfant gauche de x avec valeur v et retourner
 I8          else x ← gauche(x)
 I9      else if droit(x) = null
I10          then attacher nouvel enfant droit de x avec valeur v et retourner
I11          else x ← droit(x)
```