Cours:ASD II
Chapitre3: LES ARBRES

Dr. Adel THALJAOUI
Adel.thaljaoui@gmail.com

2017_2018

Arbres

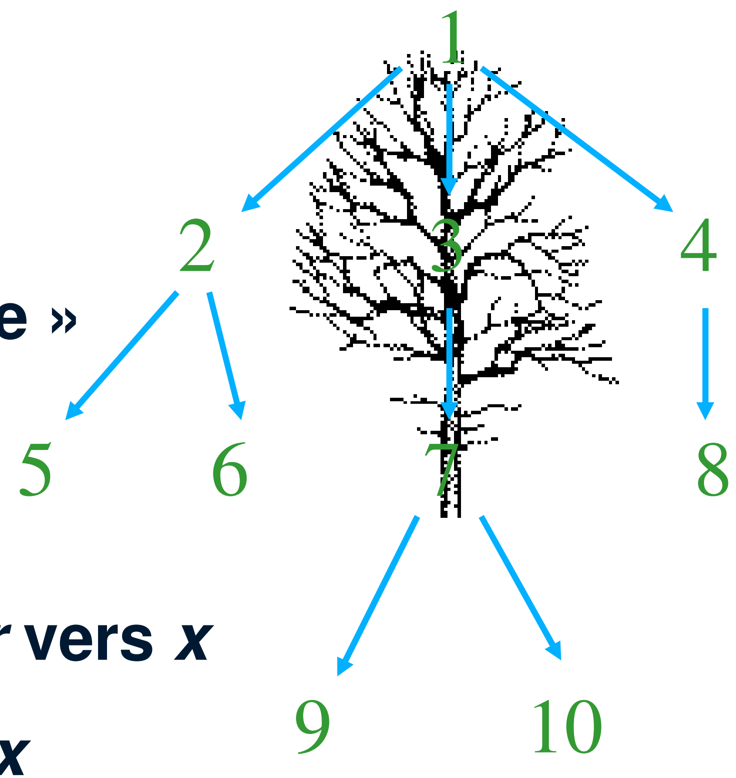
Arbre ?

Arbre ordinaire : $A = (N,P)$

- N ensemble des nœuds

- P relation binaire « parent de »

- $r \in N$ la racine



$\forall\ x \in N\ \exists$ un seul chemin de r vers x

$r = y_o\ P\ y_1\ P\ y_2\ \ldots\ P\ y_n = x$

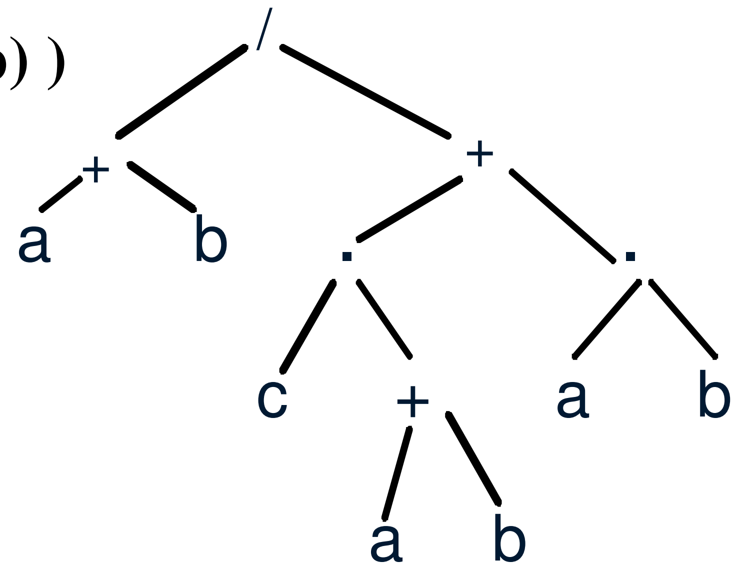
r n'a pas de parent

$\forall\ x \in N - \{\ r\ \}\quad x$ a exactement un parent

Arbres (Exemples suite)

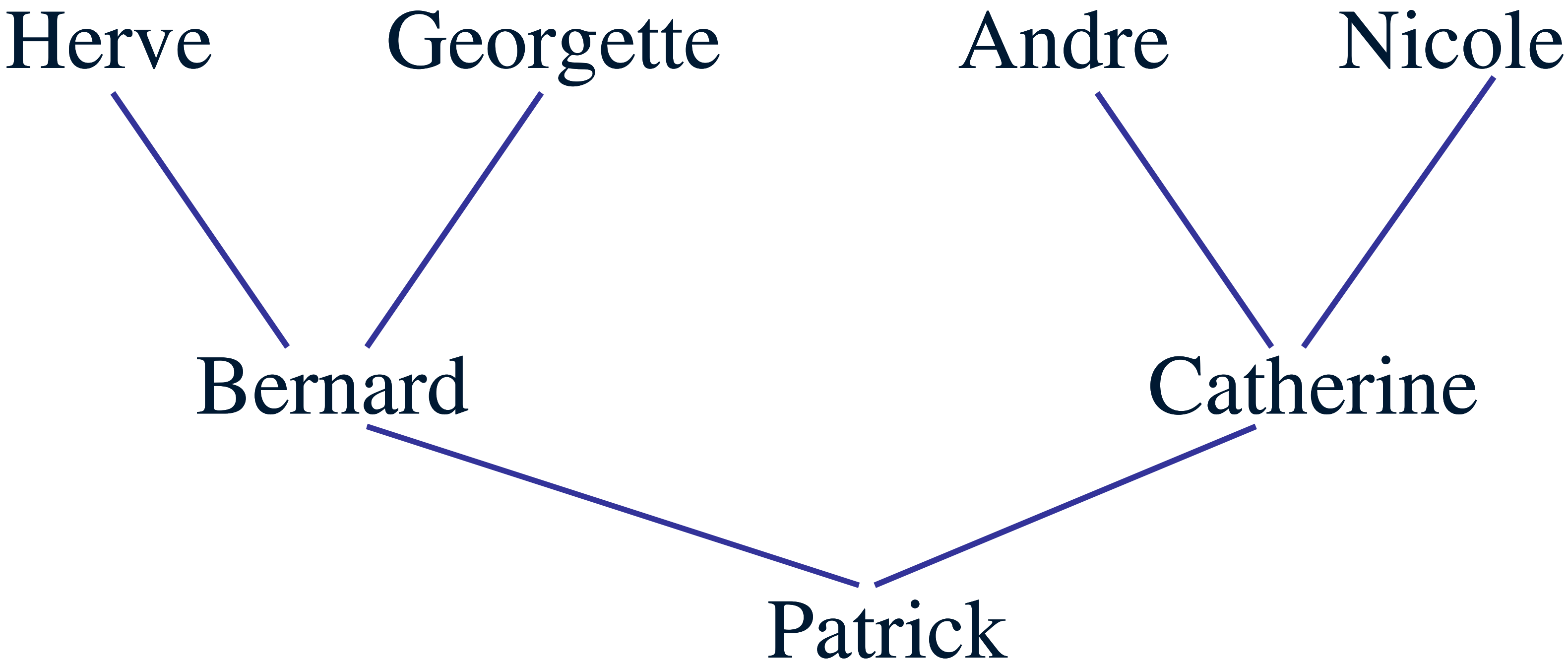
- organisation des fichiers dans des systèmes d'exploitation tels que Unix
- expression arithmétique

(a + b) / (c . (a + b) + (a . b))



Arbres (Exemples suite)

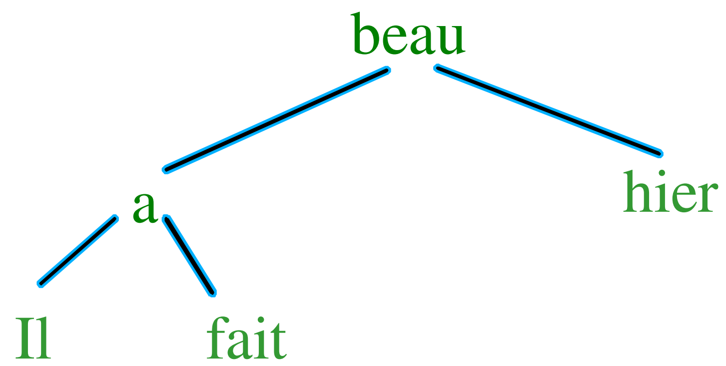
- Arbre généalogique



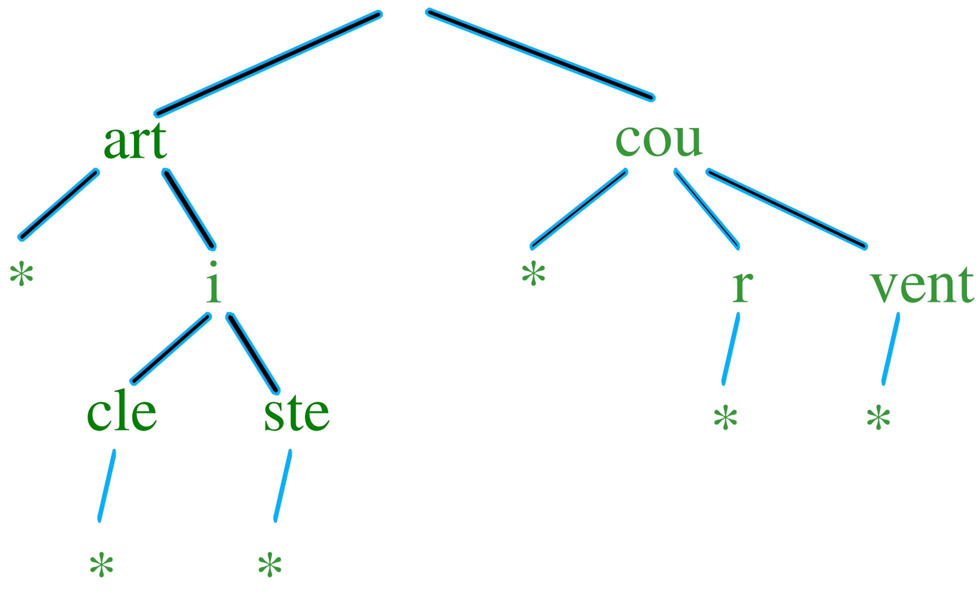
-

Arbres (Exemples suite)

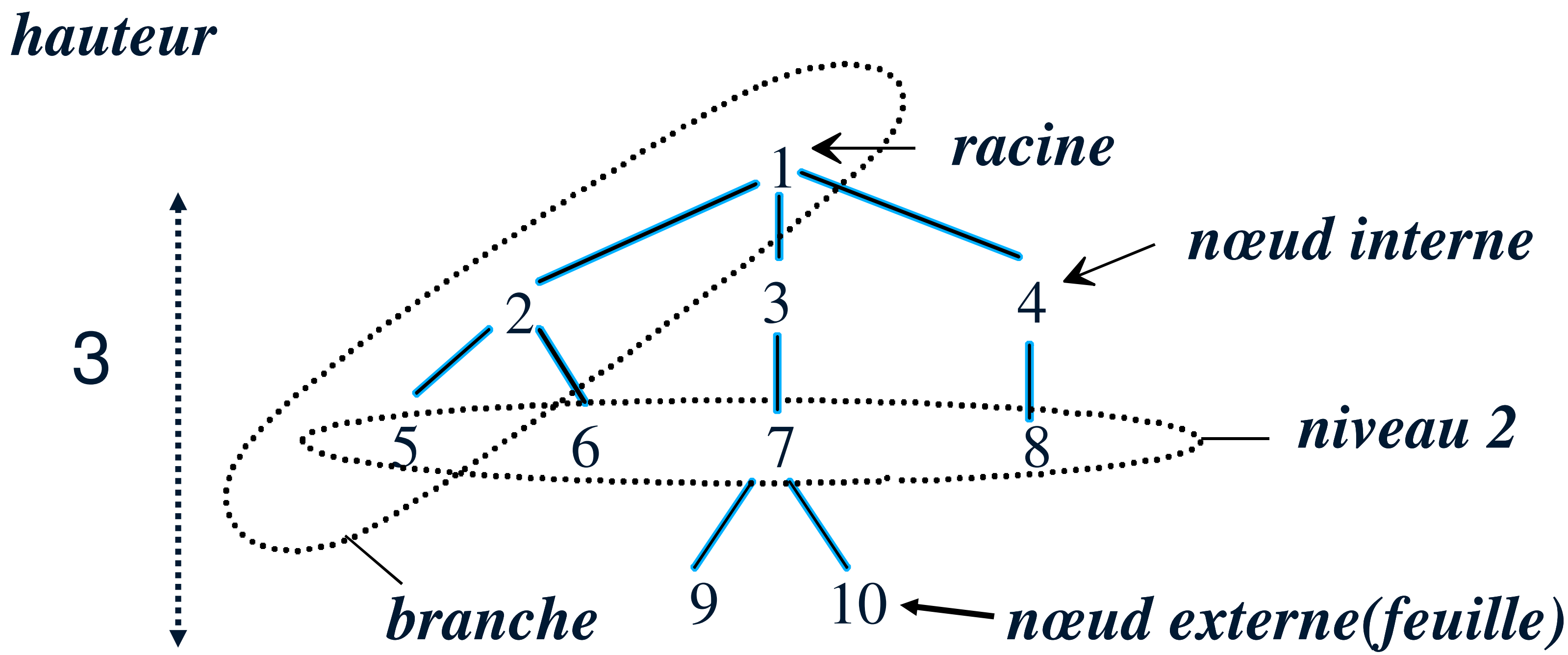
- Phrases d'une langue naturelle



- Dictionnaire



Terminologie



2, 3, 4	*enfants de 1*
3, 4	*frères de 2*
2	*fils de 1*
1	*père de 2, 3 et 4*
1, 3	*ancêtres de 7*
9, 10	*descendants de 7*

Quelques définitions

Définitions récursives

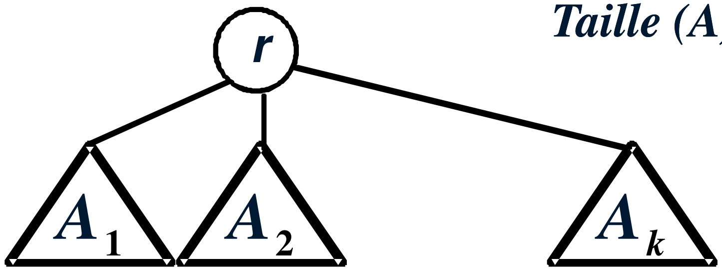
Arbre A = $\begin{cases} \Lambda & \textit{arbre vide ou} \\ (r, \{A_1, \ldots, A_k\}) & \end{cases}$

r élément, $A_1, \ldots, A_k$ *arbres*

Nœuds $(A) = \{r\} \cup (\cup \text{Nœuds}(A_i))$

Taille $(A) = |\text{Nœuds}(A)|$

$A = \Lambda$ **ou**



Une autre définition récursive

un arbre est :

- ***soit vide***
- ***soit constitué d'un nœud auquel sont chaînées un ou plusieurs sous arbres***

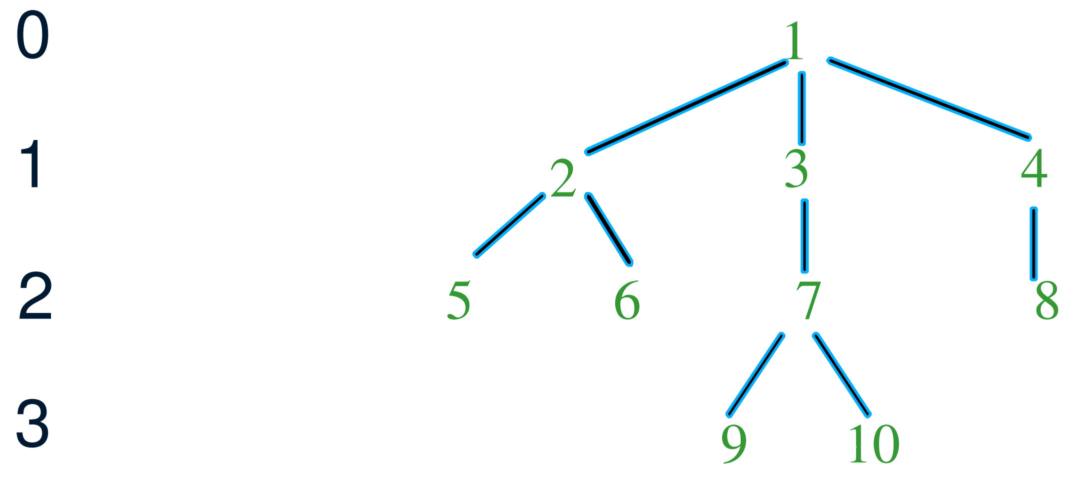
Quelques définitions

Niveaux

A arbre ***x nœud de A***

$$niveau_A(x) = \text{distance de } x \text{ à la racine}$$

$$niveau_A(x) = \begin{cases} 0 & \text{si } x = racine(A) \\ 1 + niveau\,(parent\,(x)\,) & \text{sinon} \end{cases}$$



Quelques définitions

Hauteurs

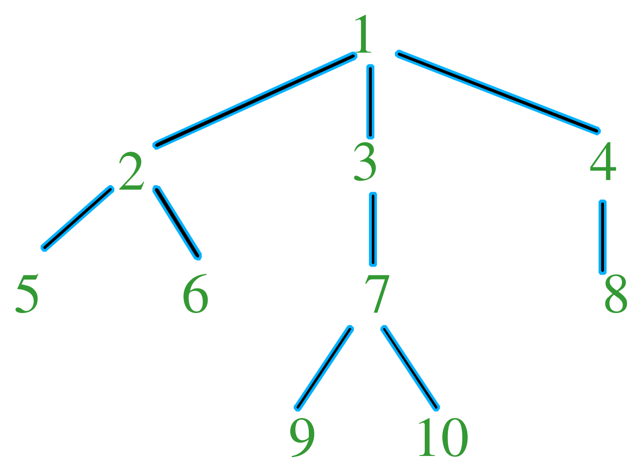
A arbre **x nœud de A**

$h_A(x)$ = distance de x à son plus lointain descendant qui est un nœud externe

$$h_A(x) = \begin{cases} 0 & \text{si A est vide} \\ 1 + \max \{ h_A(e) \mid e \text{ enfant de } x \} & \text{sinon} \end{cases}$$

$h(A) = h_A(\text{racine}(A))$

$h_A(8) = 1$
$h_A(7) = 2$
$h_A(3) = 3$
$h(A) = h_A(1) = 4$

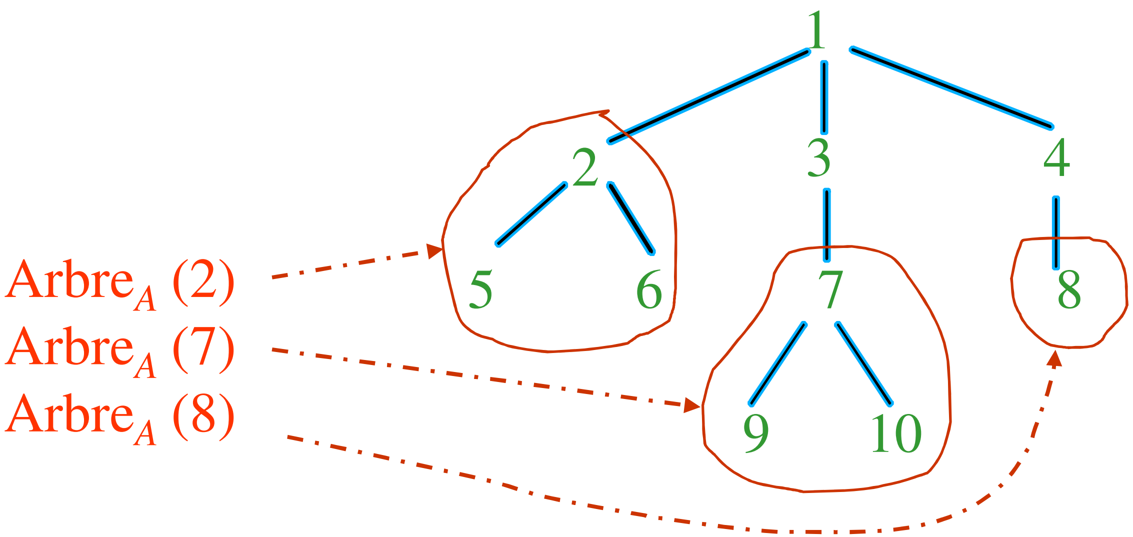


Quelques définitions

Sous-arbres

A arbre   ***x nœud de A***

$\boldsymbol{Arbre_A(x)}$ ***= sous-arbre de A qui a racine x***



Quelques définitions

Arbre binaire et arbre n-aire

- *lorsqu 'un arbre admet, pour chaque nœud, au plus n fils, l'arbre est dit n-aire*
- *si n est égale 2, l'arbre est dit binaire*

Remarque : un arbre n-aire peut être représenté par un arbre binaire équivalent

Quelques définitions

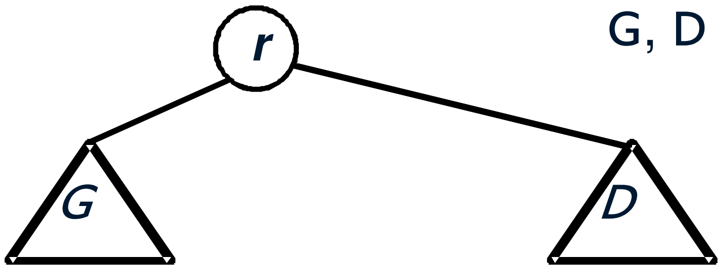
Arbre binaire : définitions récursives

Arbre binaire $A = \begin{cases} \Lambda & \textit{arbre vide} \\ \textit{ou} & \\ (r, G, D) & \end{cases}$

r élément,

G, D sous-arbres gauche et droite

$A = \Lambda$ **ou**



Une autre définition récursive

un arbre binaire est :

- ***soit vide***
- ***soit constitué d'un nœud auquel sont chaînées un sous arbre gauche et un sous arbre droit***

Arbre binaire

Représentation interne

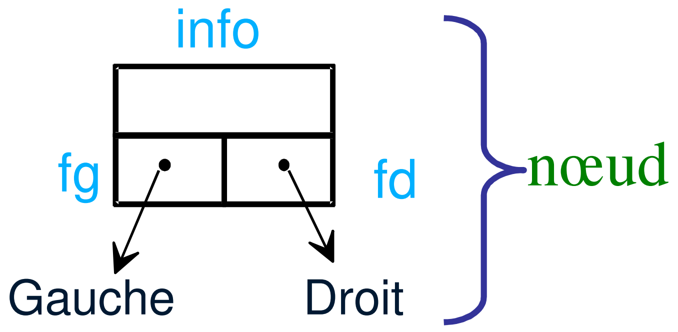
type Arbre = ^nœud;

nœud = record

info : typeElement;

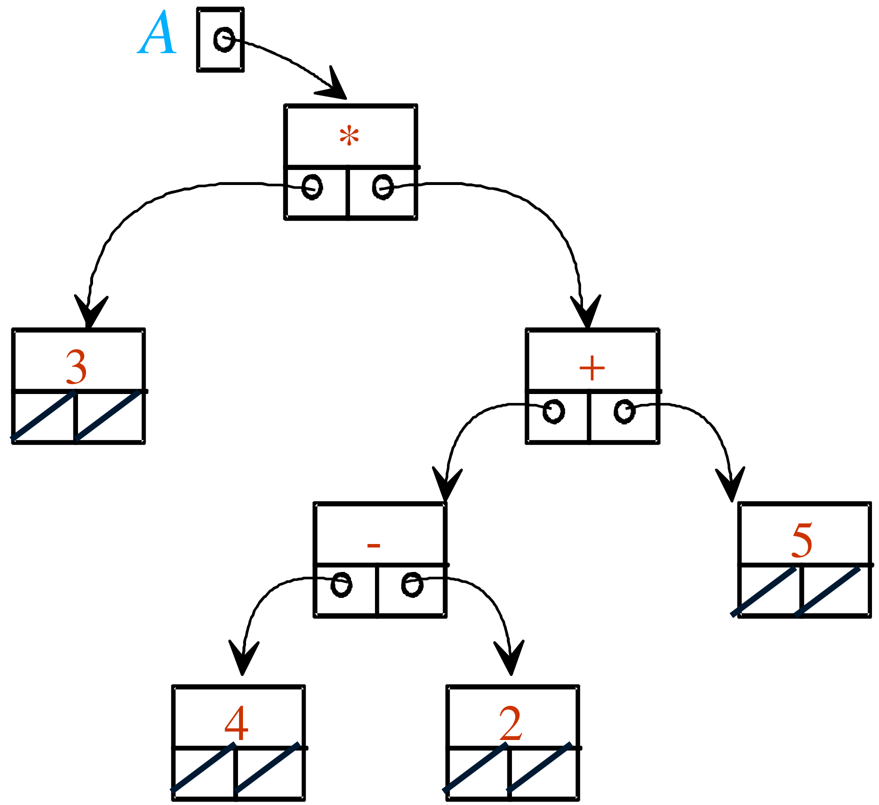
fg, fd : Arbre;

end;



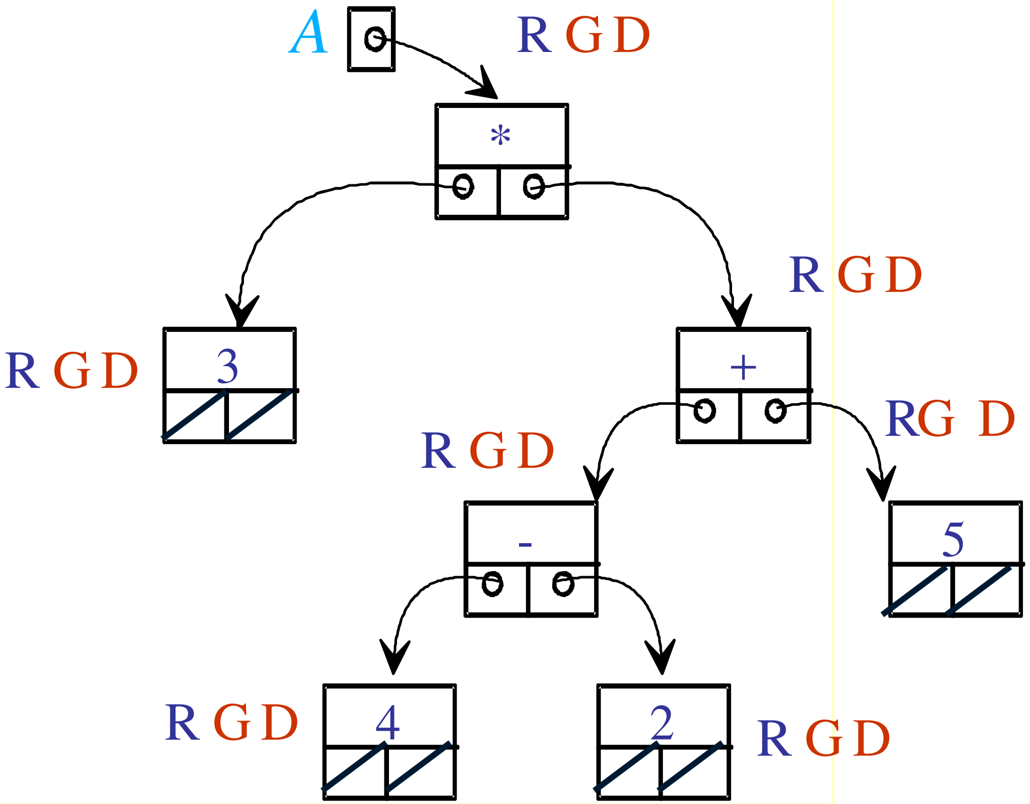
Arbre binaire (exemple)

3 * ((4 - 2) + 5)



Parcours d'un arbre binaire

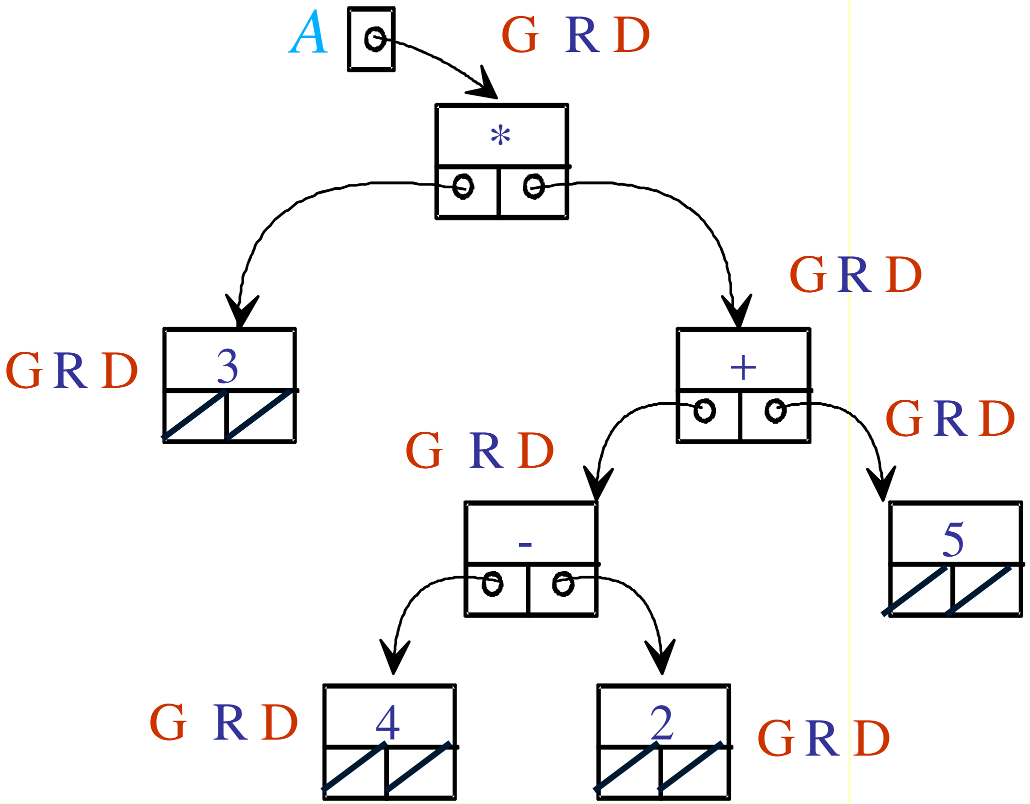
- Pré-ordre (préfixé, RGD)
  - racine
  - sous-arbre gauche
  - sous-arbre droit
- Sur l'exemple :



*** 3 + - 4 2 5**

Parcours d'un arbre binaire

- In-ordre (infixé, GRD)
  - sous-arbre gauche
  - racine
  - sous-arbre droit
- Sur l'exemple :

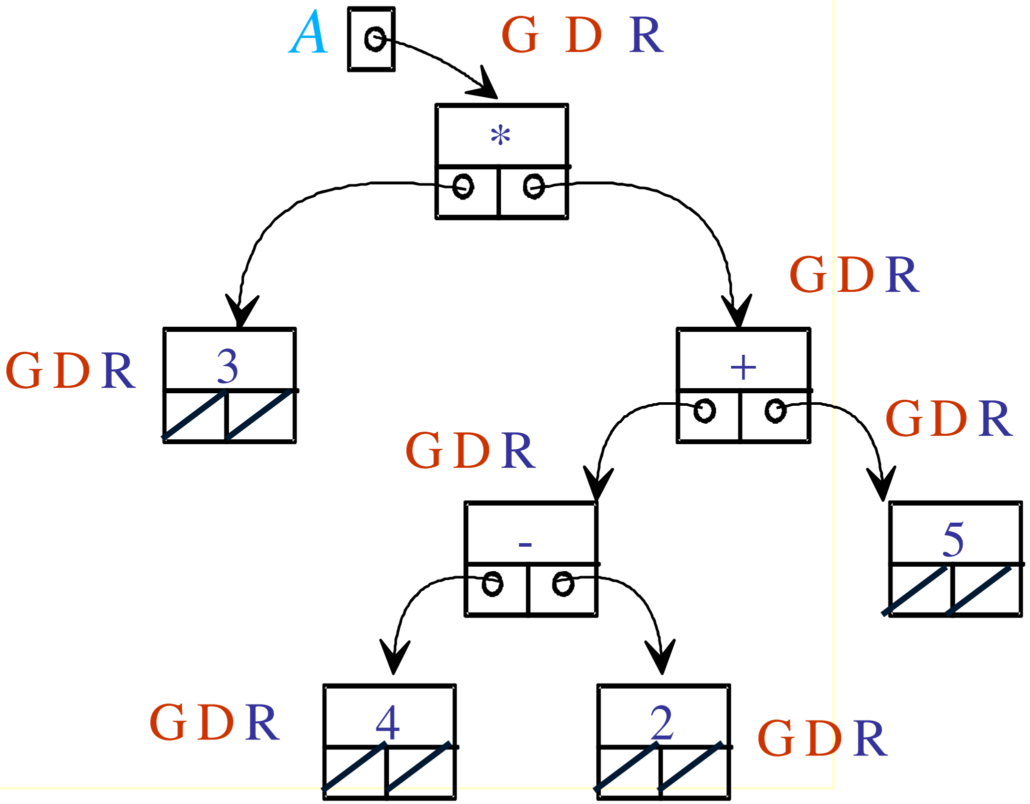


3 * 4 - 2 + 5

Parcours d'un arbre binaire

- Post-ordre (postfixé, GDR)
  - sous-arbre gauche
  - sous-arbre droit
  - racine

- Sur l'exemple :



3 4 2 - 5 + *

Algorithmes

```
procedure préfixe(a: Arbre );
debut
    si (a<>nil) alors
        ecrire(a^. info)
        préfixe(a^.fg)
        préfixe(a^.fd)
    finsi
fin
```

```
procedure infixe(a: Arbre );
debut
    si (a<>nil) alors
        infixe(a^.fg)
        ecrire(a^. info)
        infixe(a^.fd)
    finsi
fin
```

```
procedure postfixe(a: Arbre );
debut
    si (a<>nil) alors
        postfixe(a^.fg);
        postfixe(a^.fd);
        ecrire(a^. info) ;
    finsi
fin
```

Algorithmes

- **Calculer la taille d'un arbre binaire**

$$\text{Taille}(A) = \begin{cases} 0 & \text{si } A = \Lambda \quad \text{arbre vide} \\ 1 + \text{Taille}(G) + \text{Taille}(D) & \text{si } A = (r, G, D) \end{cases}$$

```
fonction taille(a: Arbre):entier
debut
        si (a=nil) alors taille= 0
        sinon
          taille ← 1+ taille(a^.fg) + taille(a^.fd)
Fin.
```

Algorithmes

- **Calcuer le nombre de feuilles d'arbre binaire**

$$\text{nbFeuilles (A)} = \begin{cases} 0 & \text{si A = nil} \quad \text{arbre vide} \\ 1 & \text{si A est une feuille} \\ \text{nbFeuille(G)+ nbFeuille(D)} & \text{si A=(}r\text{,G, D)} \end{cases}$$

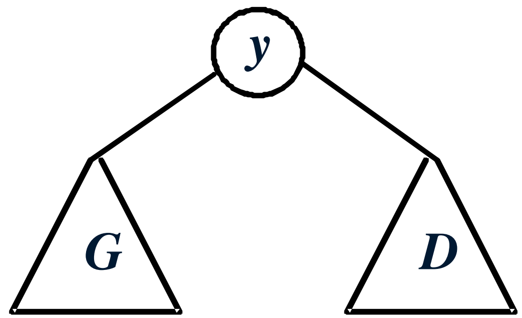
```
fonction nbFeuilles(a: Arbre): entier
debut
        if (a=nil) alors nbFeuilles = 0
        sinon if (a^.fg= nil et a^.fd=nil) alors nbFeuilles= 1
            sinon nbFeuilles ← nbFeuilles(a^.fg) + nbFeuilles(a^.fd)
fin
```

Algorithmes

Rechercher un élément

Appartient ?

Appartient (A, x) = vrai ssi x est étiquette d'un noeud de A



Appartient (A, x) =

faux	si A vide
vrai	si $x = y$
Appartient $(G(A), x)$ ou *Appartient* $(D(A), x)$;	sinon

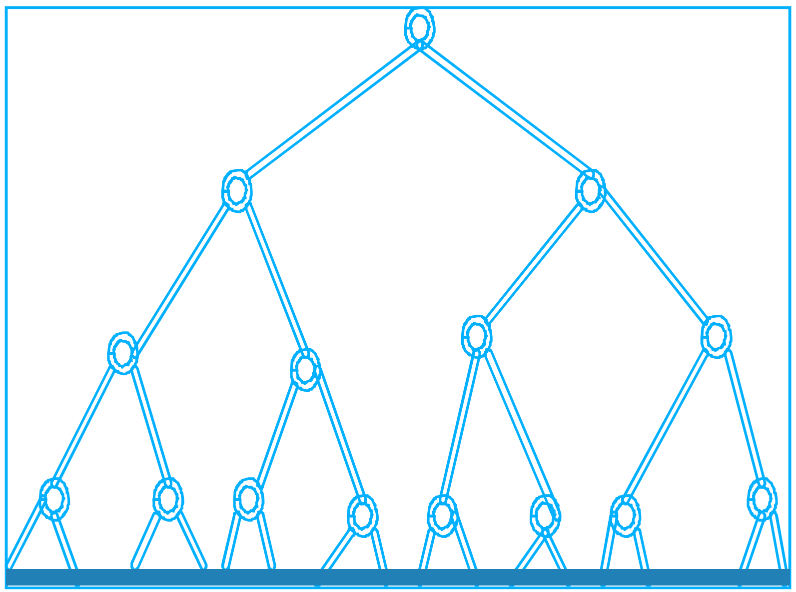
Algorithmes

```
fonction appartient(a:Arbre, val: typeElement) : boolean
debut
        if (a=nil) alors l
                appartient=faux
        sinon si (a^.info =val)
              alors appartient= true
        sinon appartient ← appartient(a->fg, val) ou
                               appartient(a->fd, val);
fin
```

Quelques définitions

Arbre binaire complet

- **chaque nœud autre qu'une feuille admet deux descendants**
- **toutes les feuilles se trouve au même niveau**



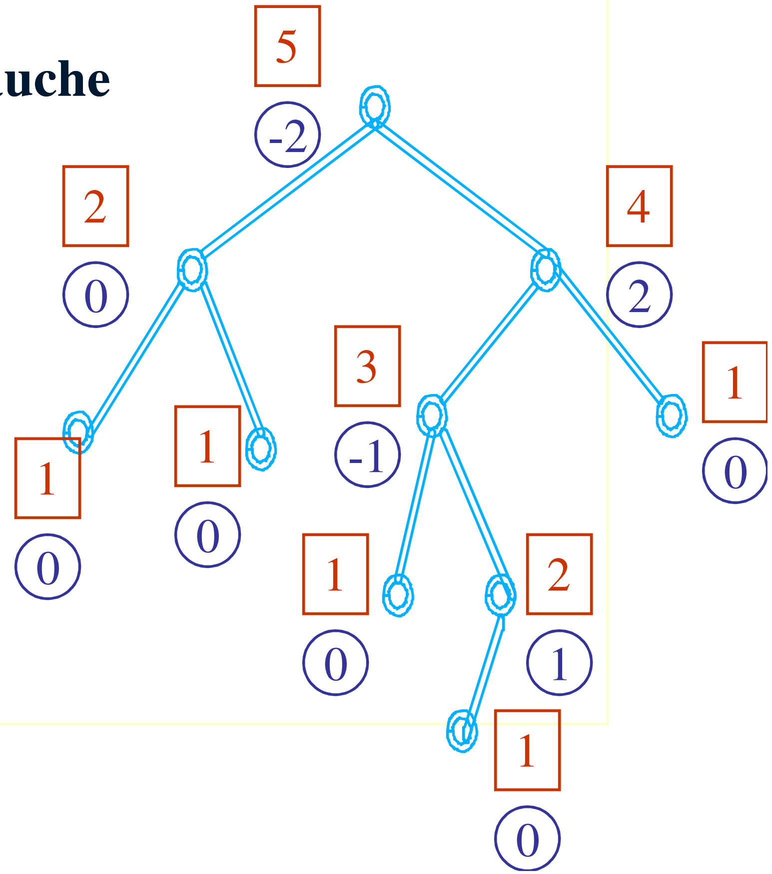
Quelques définitions

Facteur d'équilibre d 'un arbre binaire

- le facteur d'équilibre de chaque sous arbre est associé à sa racine
- le facteur d'équilibre d 'un nœud

  est égal à la hauteur du sous-arbre gauche

  moins la hauteur du sous-arbre droit



La hauteur

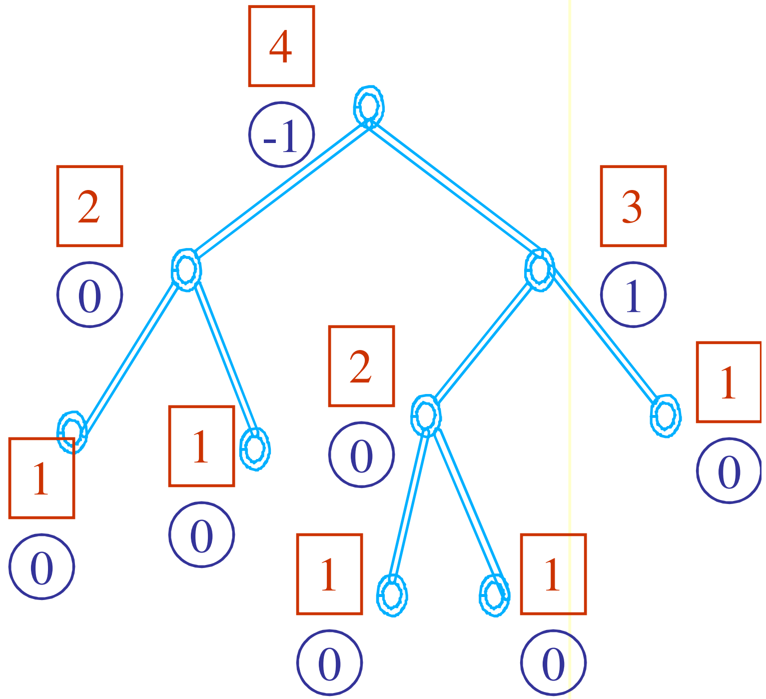
Facteur d'équilibre

Quelques définitions

arbre binaire équilibré

- **pour tout nœud de l'arbre la valeur absolue du facteur d 'équilibre est inférieur ou égale à 1**

Arbre binaire équilibré →



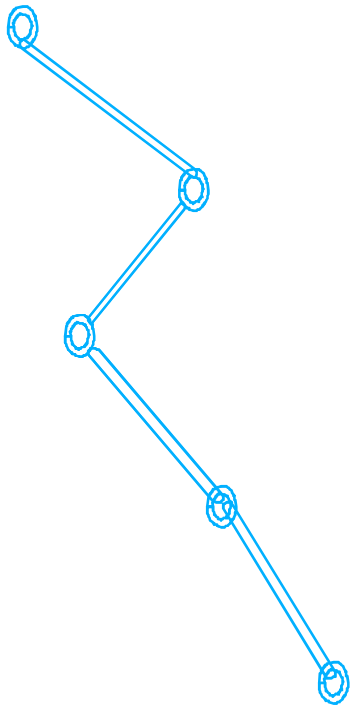
□ **La hauteur**

○ **Facteur d'équilibre**

Quelques définitions

Arbre binaire dégénéré

- **un arbre binaire est dit dégénéré, si tous les nœuds de cet arbre ont au plus 1 fils.**



Quelques définitions

Arbre binaire de ordonnée (de recherche)

Soit A un arbre binaire

nœuds étiquetés par des éléments

A est un arbre binaire ordonnée (de recherche) :

ssi en tout noeud p de A

$$Elt(G(p)) \leq Elt(p) < Elt(D(p))$$

ssi A = Arbre_vide ou

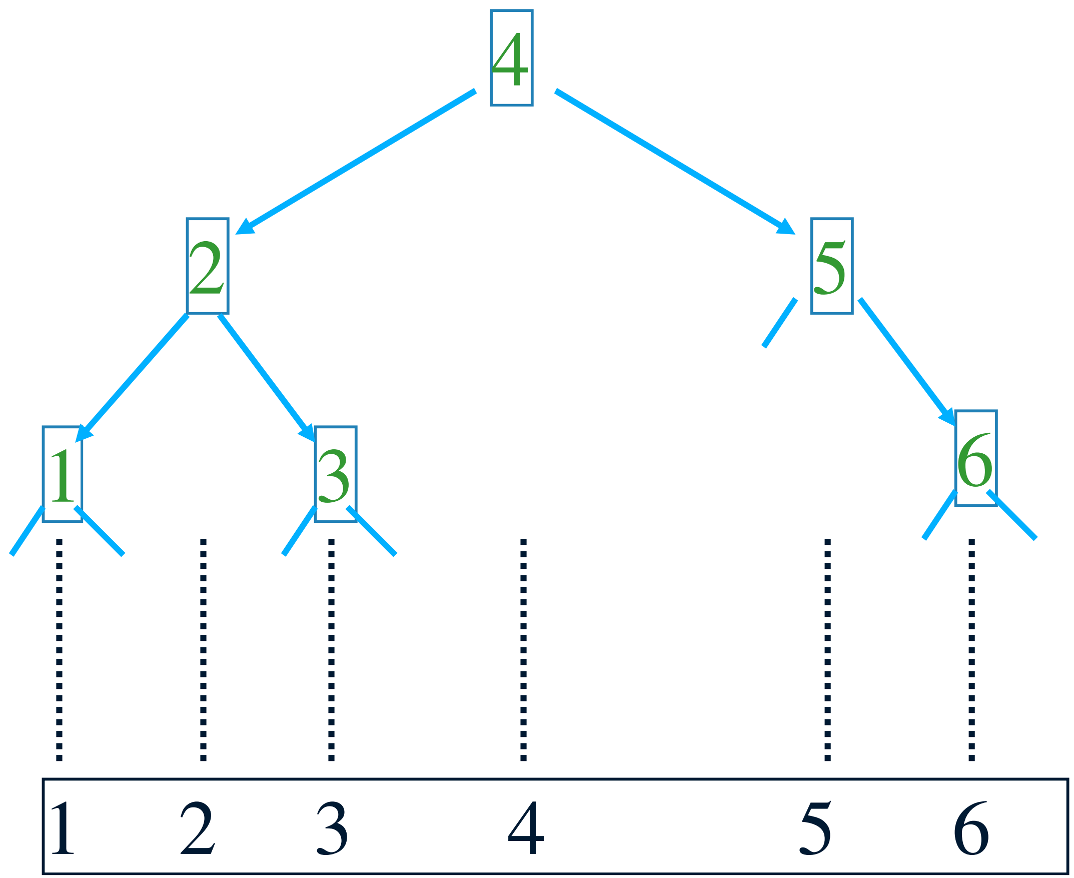
A = (r, G, D) avec

. G, D arbres binaires de recherche et

. Elt(G) ≤ Elt (r) < Elt(D)

Arbre binaire de recherche

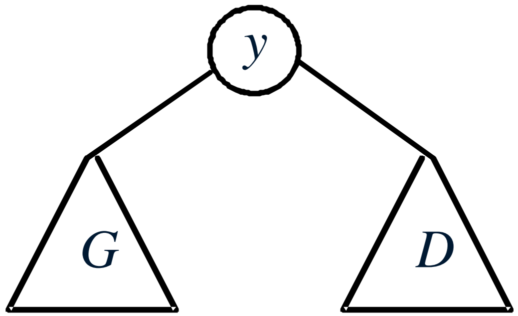
Exemple



Arbre binaire de recherche

Appartient ?

Appartient (A, x) ***= vrai ssi*** x ***est étiquette d'un noeud de*** A



Appartient $(A, x) =$

faux	***si*** A ***vide***
vrai	***si*** $x = y$
Appartient $(G(A), x)$	***si*** $x < y$
Appartient $(D(A), x)$	***si*** $x > y$

Arbre binaire de recherche

- *Version récursive*

```
fonction appartient(a:Arbre, val:typeElement) :boolean
debut
    si (a=nil) alors
     appartient ← faux
    sinon si (a^.info = val) alors
            appartient ← true
           sinon si (val < a^.info) alors
             appartient ← appartient(a^.fg, val)
                 sinon appartient ← appartient(a^.fd, val)
Fin.
```

Arbre binaire de recherche

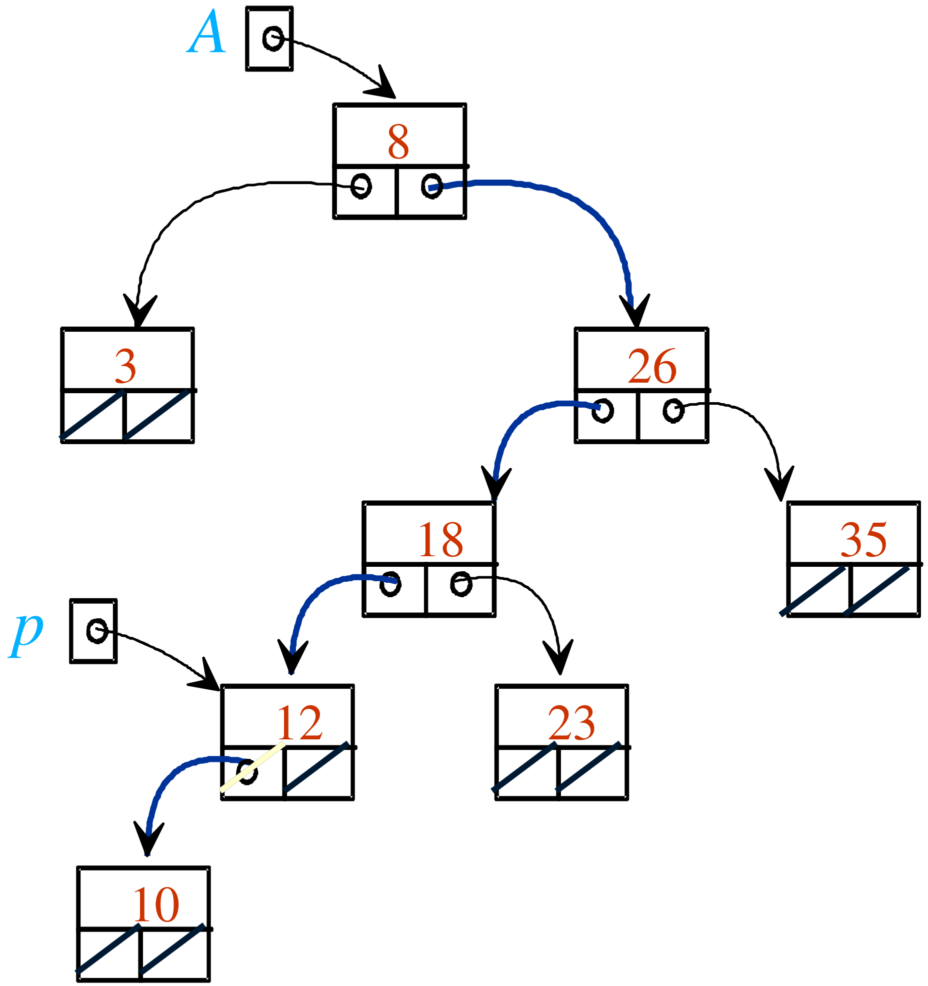
- *Version itérative*

```
fontion  appartient(a:Arbre, val:typeElement) : boolean
    variable trouve:boolean
debut
    trouve ← false
    Tant que (a<>nil and (trouve= faux)
    Faire
            trouve ← (a^.info = val)
            si (val < a^.info) alors
             a ← a^.fg
            sinon a ← a^.fd
            Finsi
FinFaire
    appartient ← trouve
Fin
```

Arbre binaire de recherche

- *Ajout d'une valeur donnée*

Val* = *10



Arbre binaire de recherche

- *Version itérative*

```
procedure ajout (var a:Arbre,
    val:integer)
variable
   p , pere:Arbre
debut
    p ← a
   pere ← nil
TantQue(p<>nil)
Faire
    pere ← p;
    si(val <= p^.info) alors
            p ← p^.fg
    else
        p ← p^.fd;
Finsi
FintanQue
allouer(p)
 p^.info ← val
 p^.fg ← nil
 p^.fd ← nil
    si (pere =nil) alors
        a ← p
    sinon
     si(val <=pere^.info)alors
          pere^.fg ← p
          sinon
        pere^.fd ← p
end;
```

Arbre binaire de recherche

- *Version récursive*

```
procedure ajout (var a:Arbre, val:integer)
si(a =nil)  alors
    allouer(a)
    a^.info ← val
    a^.fg ← nil
    a^.fd ← nil;

Sinon si (val <= a^.info) alors
                    ajout (a^.fg, val)
          sinon ajout (a^.fd, val)
Fin
```