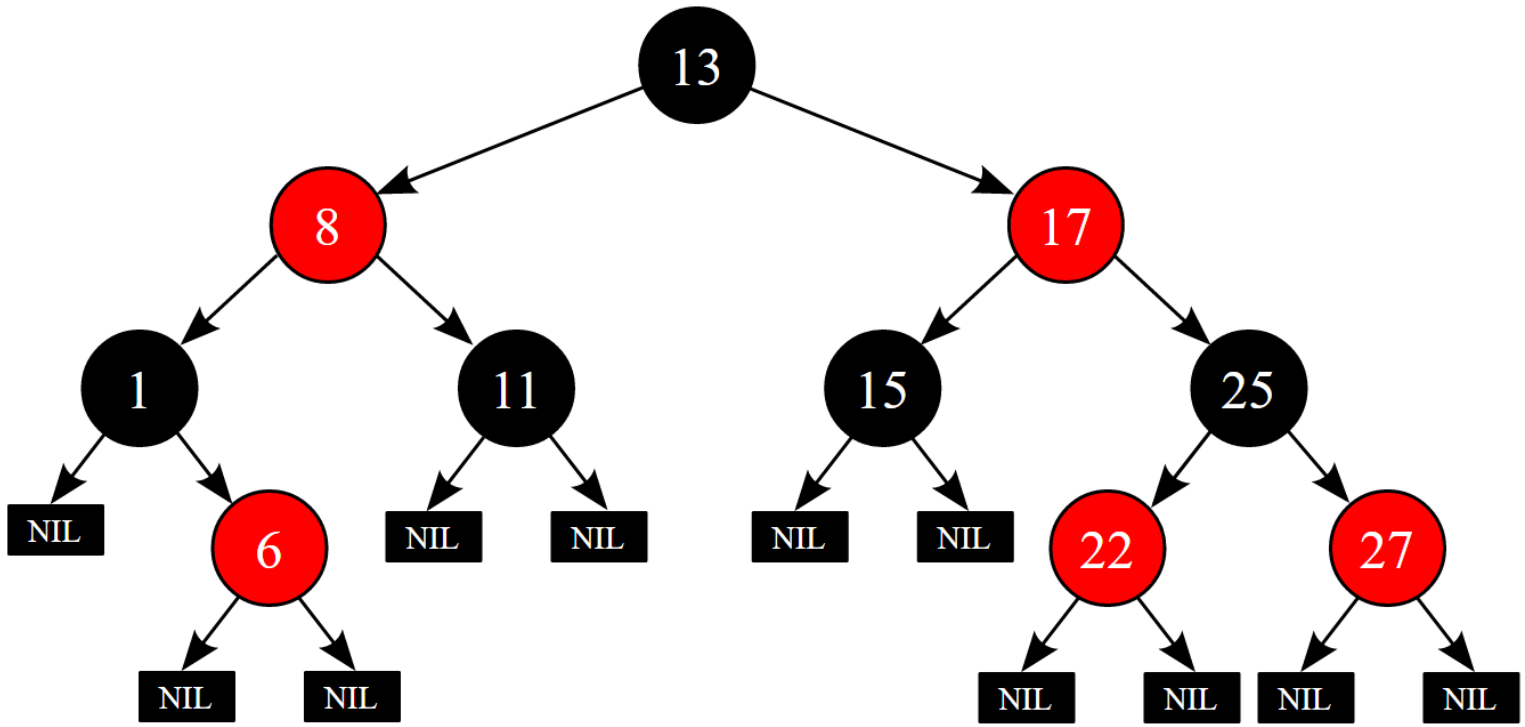
Chapitre 7 : Arbres



Arbres : def. mathématique

Définition mathématique

Arbre A ensemble fini, non vide muni d'une relation binaire $\prec$ t.q :

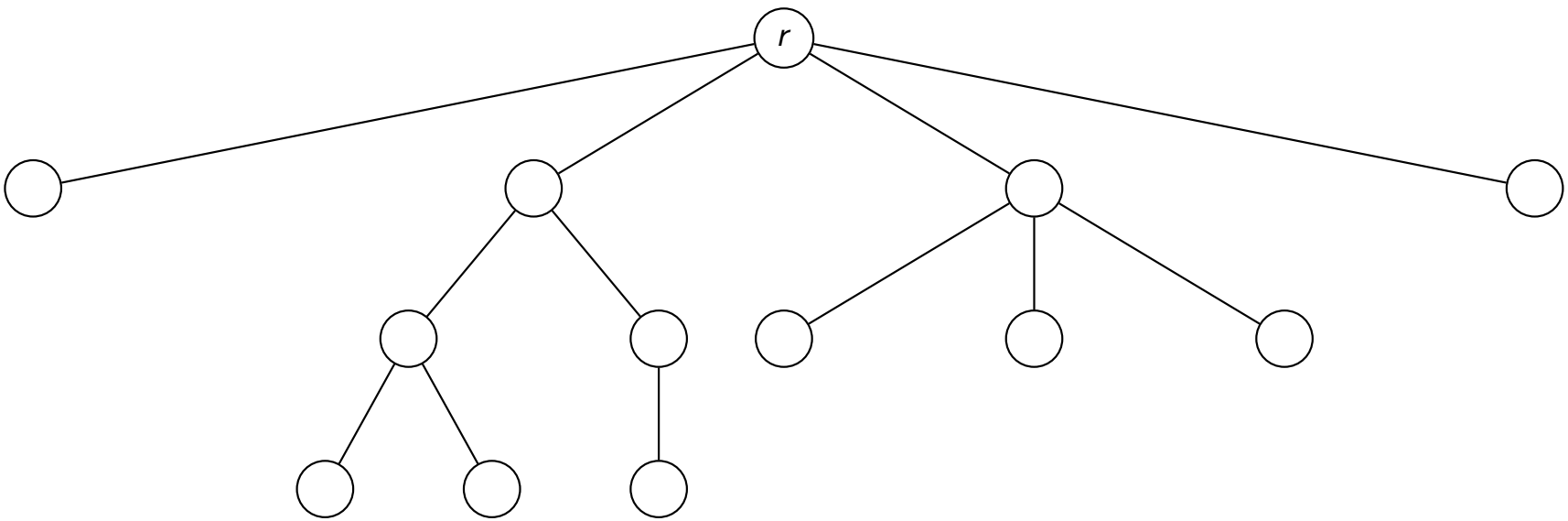
- $\exists! r \in A \quad \forall x \in A \quad \neg(r \prec x)$. L'élément r s'appelle la racine de l'arbre.
- $\forall x \in A \backslash \{r\} \quad \exists! t \in A \quad x \prec y$. On dit que y est *le* parent (ou père) de x, et x est *un* fils de y.
- $\forall x \in A \backslash \{r\} \quad \exists n > 0 \quad \exists (x_1, \ldots, x_n) \in A^n \quad x \prec x_1 \prec x_2 \prec \cdots \prec x_n = r$.

Arbres : def. mathématique

Définition mathématique

Arbre A ensemble fini, non vide muni d'une relation binaire $\prec$ t.q :

- $\exists! r \in A \quad \forall x \in A \quad \neg(r \prec x)$. L'élément r s'appelle la racine de l'arbre.
- $\forall x \in A \backslash \{r\} \quad \exists! t \in A \quad x \prec y$. On dit que y est *le* parent (ou père) de x, et x est *un* fils de y.
- $\forall x \in A \backslash \{r\} \quad \exists n > 0 \quad \exists (x_1, \ldots, x_n) \in A^n \quad x \prec x_1 \prec x_2 \prec \cdots \prec x_n = r$.



Arbres : arité, feuilles, nœuds internes

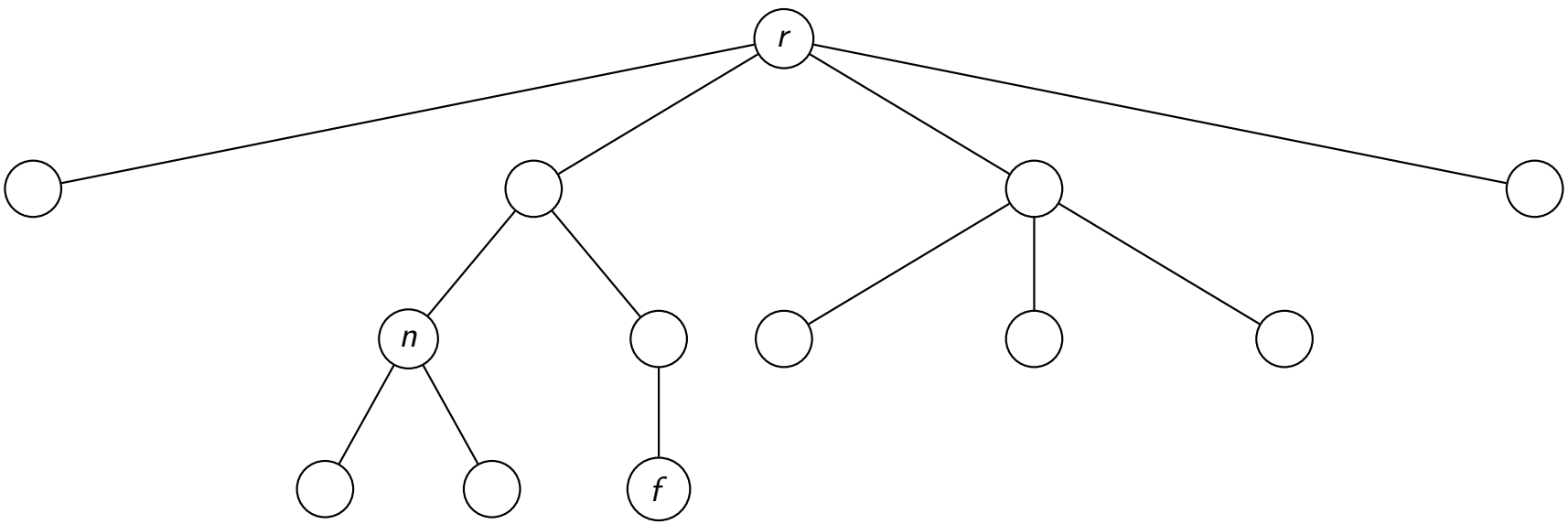
Définitions

- arité d'un nœud : son nombre de fils ;
- nœud interne : arité > 0
- feuille : arité $= 0$;

Arbres : arité, feuilles, nœuds internes

Définitions

- arité d'un nœud : son nombre de fils ;
- nœud interne : arité > 0
- feuille : arité $= 0$;



Arbres : arbres binaires, binaires entiers

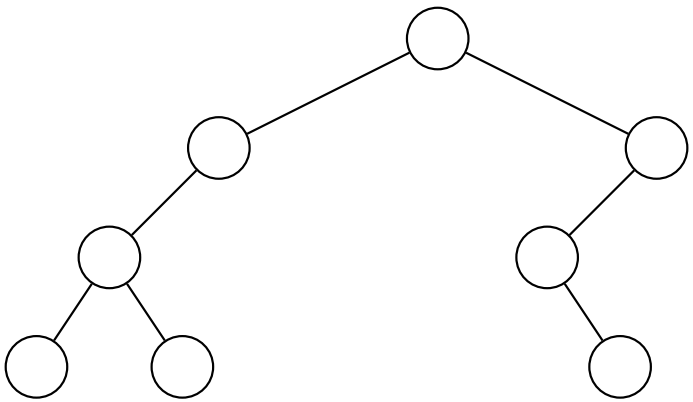
Définitions

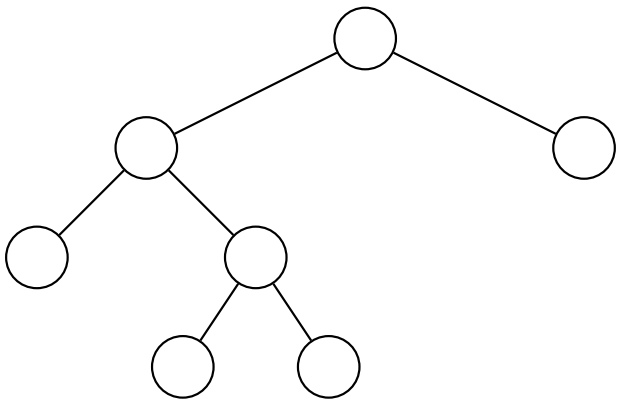
- arbre binaire : arité ≤ 2 pour tout nœud ;
- arbre binaire entier (strict) : arité 0 ou 2 pour tout nœud ;

Arbres : arbres binaires, binaires entiers

Définitions

- arbre binaire : arité ≤ 2 pour tout nœud ;
- arbre binaire entier (strict) : arité 0 ou 2 pour tout nœud ;





Arbres : profondeur / hauteur

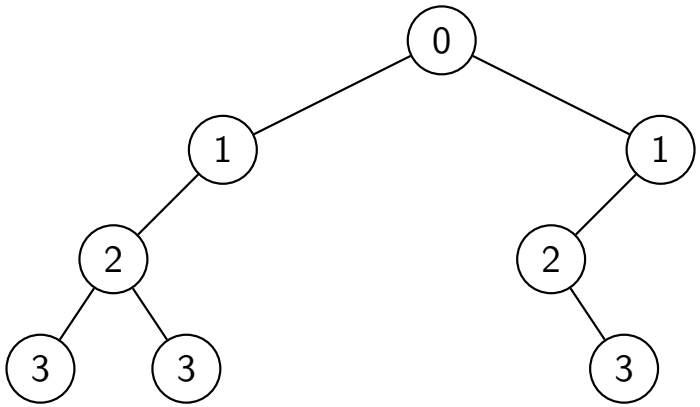
Définitions

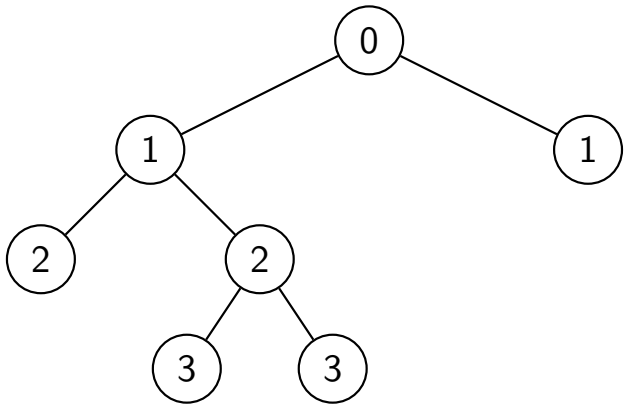
- avec r racine de l'arbre et x nœud : $\exists\,! n \geq 0, \exists\,! x_1, \ldots, x_{n-1}$ tels que $x \prec x_1 \prec \cdots \prec x_{n-1} \prec x_n = r$.
- Entier n : *profondeur* de x.
- hauteur de l'arbre = profondeur maximale de ses nœuds.

Arbres : profondeur / hauteur

Définitions

- avec r racine de l'arbre et x nœud : $\exists\,! n \geq 0, \exists\,! x_1, \ldots, x_{n-1}$ tels que $x \prec x_1 \prec \cdots \prec x_{n-1} \prec x_n = r$.
- Entier n : *profondeur* de x.
- hauteur de l'arbre = profondeur maximale de ses nœuds.





Encadrement hauteur / nombre de nœuds

Prop.

A de hauteur h, arité max. a. Si $a > 1$:

$$h + 1 \leq n \leq \frac{a^{h+1} - 1}{a - 1}$$

Encadrement hauteur / nombre de nœuds

Prop.

A de hauteur h, arité max. a. Si $a > 1$:

$$h + 1 \leq n \leq \frac{a^{h+1} - 1}{a - 1}$$

$\rightsquigarrow$ Preuve.

Encadrement hauteur / nombre de nœuds

Prop.

A de hauteur h, arité max. a. Si $a > 1$:

$$h + 1 \leq n \leq \frac{a^{h+1} - 1}{a - 1}$$

$\rightsquigarrow$ Preuve.

Corollaire

Hauteur h d'un arbre à n nœuds tous d'arité au plus $a > 1$ vérifie :

$$\log_a((a-1)n + 1) - 1 \leq h \leq n - 1$$

Encadrement hauteur / nombre de nœuds

Prop.

A de hauteur h, arité max. a. Si $a > 1$:

$$h + 1 \leq n \leq \frac{a^{h+1} - 1}{a - 1}$$

$\rightsquigarrow$ Preuve.

Corollaire

Hauteur h d'un arbre à n nœuds tous d'arité au plus $a > 1$ vérifie :

$$\log_a((a-1)n + 1) - 1 \leq h \leq n - 1$$

Corollaire

Soit A un arbre binaire à n nœuds. Sa hauteur h vérifie
$\lfloor \log_2(n) \rfloor \leq h \leq n - 1$

Encadrement hauteur / nombre de nœuds

Prop.

A de hauteur h, arité max. a. Si $a > 1$:

$$h + 1 \leq n \leq \frac{a^{h+1} - 1}{a - 1}$$

$\rightsquigarrow$ Preuve.

Corollaire

Hauteur h d'un arbre à n nœuds tous d'arité au plus $a > 1$ vérifie :

$$\log_a((a-1)n + 1) - 1 \leq h \leq n - 1$$

Corollaire

Soit A un arbre binaire à n nœuds. Sa hauteur h vérifie
$\lfloor \log_2(n) \rfloor \leq h \leq n - 1$

$\rightsquigarrow$ Preuve : $h \geq \log_2(n+1) - 1 > \log_2(n) - 1 \geq \lfloor \log_2(n) \rfloor - 1$

Feuilles et nœuds internes d'un arbre binaire

Prop.

Un arbre binaire entier à p nœuds internes a $p+1$ feuilles.

Feuilles et nœuds internes d'un arbre binaire

Prop.

Un arbre binaire entier à p nœuds internes a $p+1$ feuilles.

$\rightsquigarrow$ Preuve : récurrence. $(n_g + 1) + (n_d + 1) = (n_g + n_d + 1) + 1$.

Feuilles et nœuds internes d'un arbre binaire

Prop.

Un arbre binaire entier à p nœuds internes a $p+1$ feuilles.

$\rightsquigarrow$ Preuve : récurrence. $(n_g+1)+(n_d+1)=(n_g+n_d+1)+1$.

Corollaire

Un arbre binaire entier à p nœuds internes a au plus $p+1$ feuilles.

Feuilles et nœuds internes d'un arbre binaire

Prop.

Un arbre binaire entier à p nœuds internes a $p+1$ feuilles.

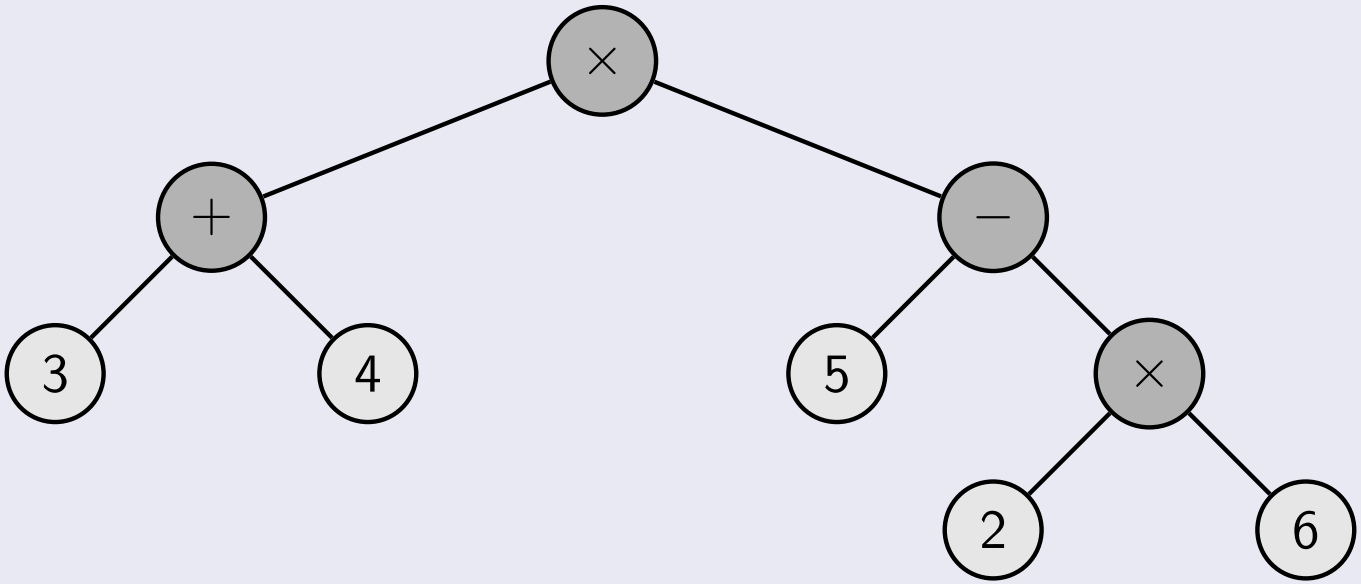
$\rightsquigarrow$ Preuve : récurrence. $(n_g+1)+(n_d+1)=(n_g+n_d+1)+1$.

Corollaire

Un arbre binaire entier à p nœuds internes a au plus $p+1$ feuilles.

$\rightsquigarrow$ Preuve. Rajouter un fils (feuille) à chaque nœud d'arité 1.

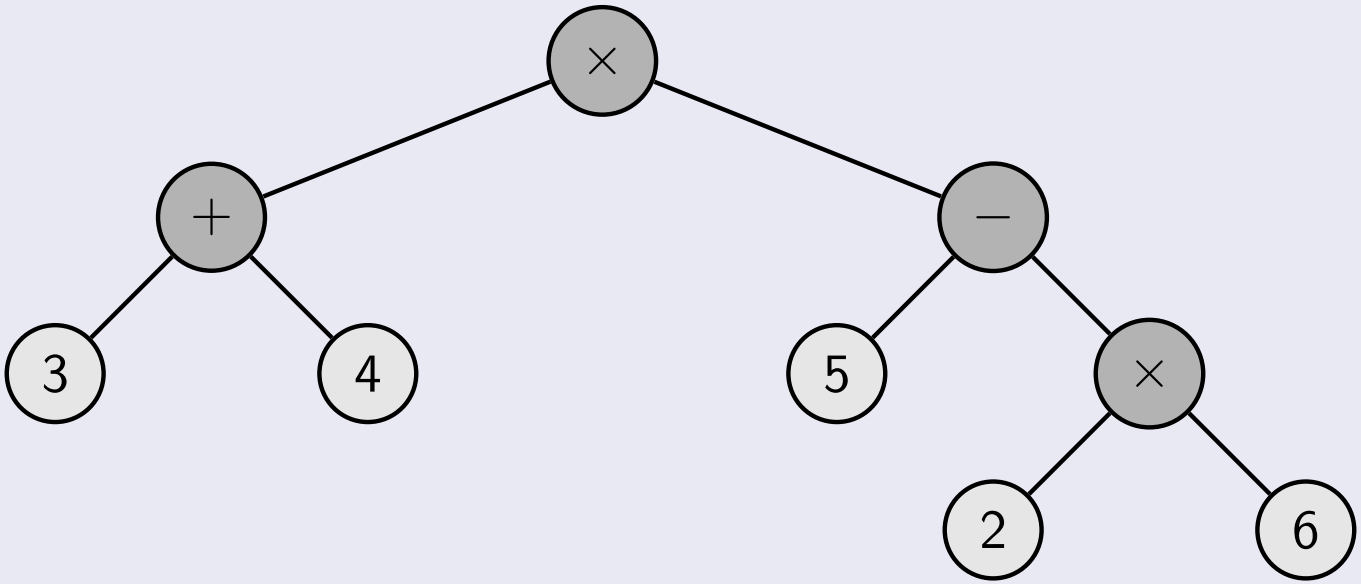
Arbres en informatique



Arbre associé à $(3 + 4) \times (5 - (2 \times 6))$

Arbres en Caml

Arbres en informatique

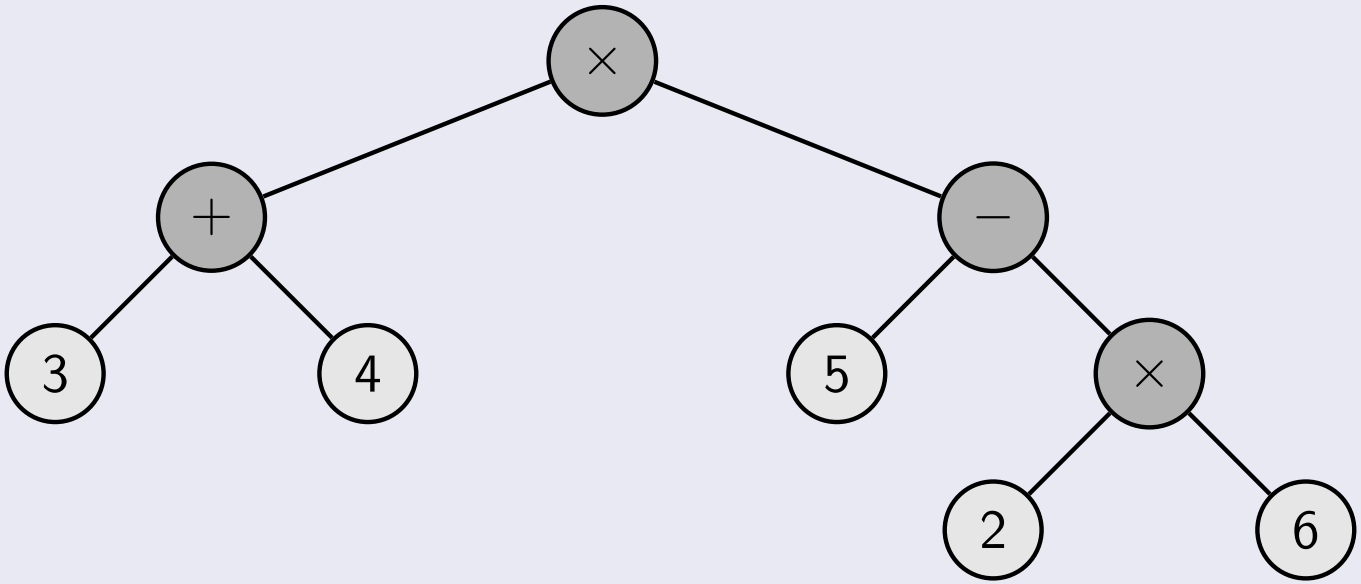


Arbre associé à $(3 + 4) \times (5 - (2 \times 6))$

- ordonnancement des fils d'un nœud ;
- les nœuds portent des étiquettes.

Arbres en Caml

Arbres en informatique



Arbre associé à $(3 + 4) \times (5 - (2 \times 6))$

- ordonnancement des fils d'un nœud ;
- les nœuds portent des étiquettes.

Choix d'implémentation

- Persistante : à la manière des listes chaînées
- Mutable : l'an prochain un peu.

Implémentation des arbres généraux

Idée

arbre = étiquette + liste sous-arbres (éventuellement vide)

Implémentation des arbres généraux

Idée

arbre = étiquette + liste sous-arbres (éventuellement vide)

Type

```
type 'a arbre = N of 'a * 'a arbre list;;
```

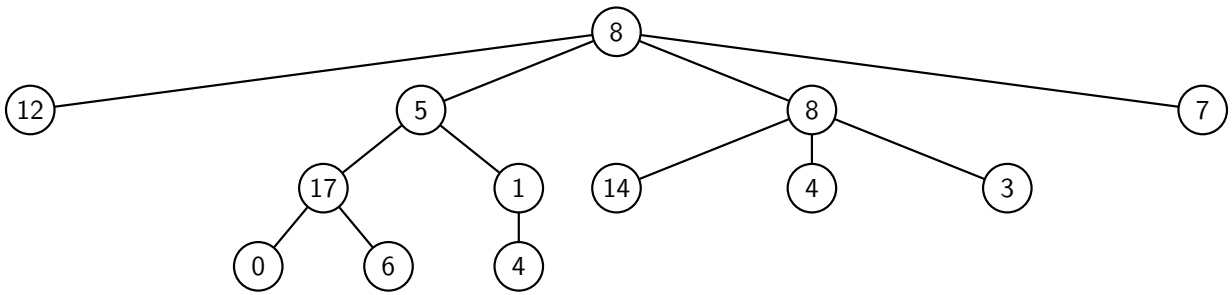
Implémentation des arbres généraux

Idée

arbre = étiquette + liste sous-arbres (éventuellement vide)

Type

```
type 'a arbre = N of 'a * 'a arbre list;;
```




```
#ex_arbre ;;
- : int arbre =
N (8,
 [N (12, []);
  N (5, [N (17, [N (0, []); N (6, [])]); N (1, [N (4, [])])]);
  N (8, [N (14, []); N (4, []); N (3, [])]);
  N (7, [])])
```

Exemple de fonction

Hauteur

```
let rec hauteur a=match a with
  | N(_,q) -> 1+max_h q
and max_h q=match q with
  | [] -> -1
  | x::p -> max (hauteur x) (max_h p)
;;
```

```
# hauteur ex_arbre ;;
- : int = 3
```

Arbres binaires entiers

Idée

arbre = feuille ou étiquette + 2 sous-arbres

Arbres binaires entiers

Idée

arbre = feuille ou étiquette + 2 sous-arbres

Type

```
type ('a, 'b) arbre =
       F of 'a | N of 'b*('a, 'b) arbre * ('a, 'b) arbre;;
```

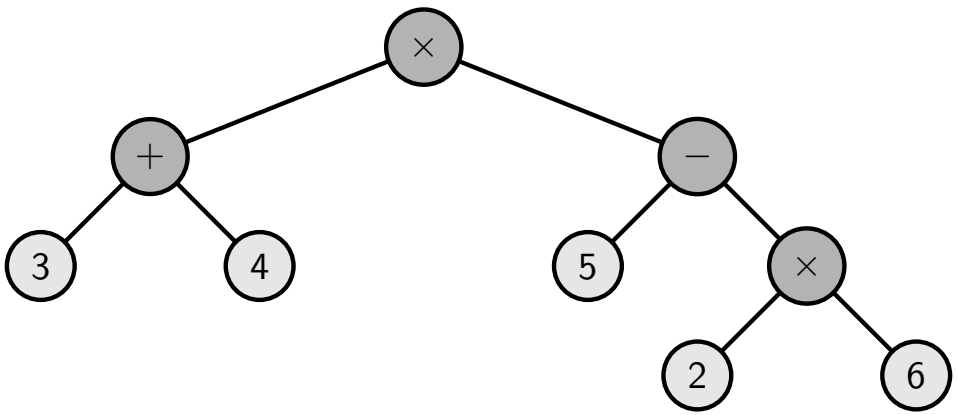
Arbres binaires entiers

Idée

arbre = feuille ou étiquette + 2 sous-arbres

Type

```
type ('a, 'b) arbre =
       F of 'a | N of 'b*('a, 'b) arbre * ('a, 'b) arbre;;
```



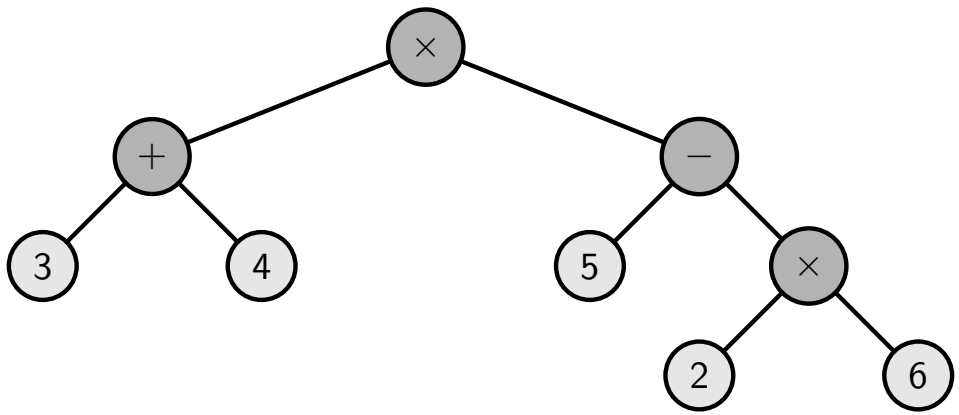
Arbres binaires entiers

Idée

arbre = feuille ou étiquette + 2 sous-arbres

Type

```
type ('a, 'b) arbre =
       F of 'a | N of 'b*('a, 'b) arbre * ('a, 'b) arbre;;
```




```
type op = Plus | Moins | Fois | Div | Mod ;;
let expr = N (Fois, N (Plus, F 3, F 4),
      N (Moins, F 5, N (Fois, F 2, F 6))) ;;
```

Exemple de fonction

```
let rec evalue e=match e with
  | F x -> x
  | N (op, a, b) -> (traduit op) (evalue a) (evalue b)
;;
```

Exemple de fonction

```
let rec evalue e=match e with
  | F x -> x
  | N (op, a, b) -> (traduit op) (evalue a) (evalue b)
;;

# evalue ;;
- : (int, op) arbre -> int = <fun>
# evalue expr ;;
- : int = -49
```

Arbres binaires

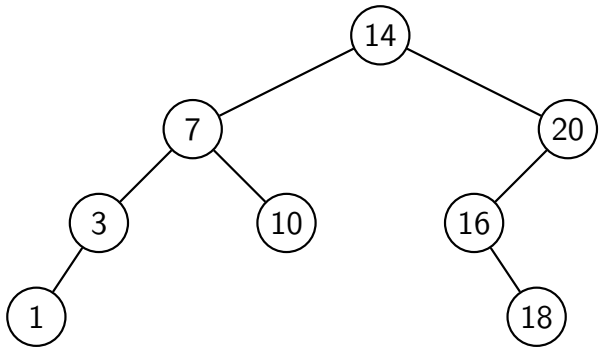
Idée

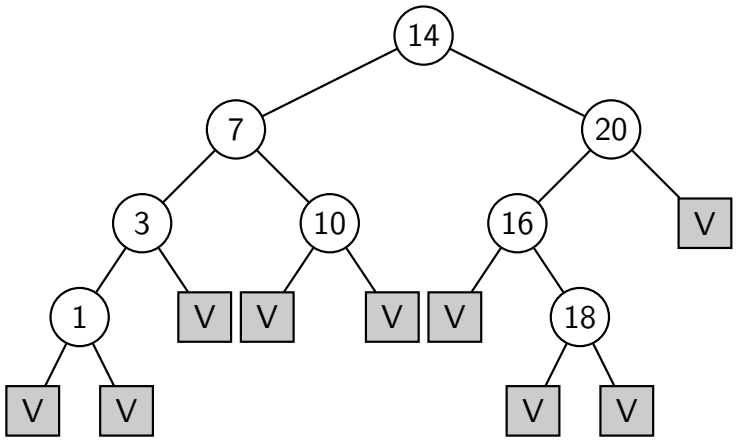
arbre = vide (rien) ou étiquette + 2 sous-arbres

Arbres binaires

Idée

arbre = vide (rien) ou étiquette + 2 sous-arbres

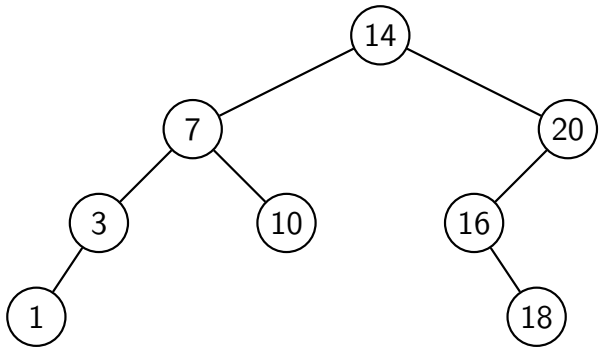


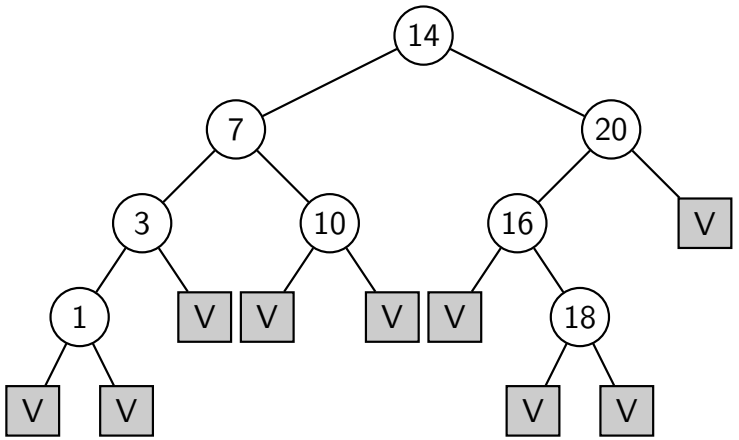


Arbres binaires

Idée

arbre = vide (rien) ou étiquette + 2 sous-arbres





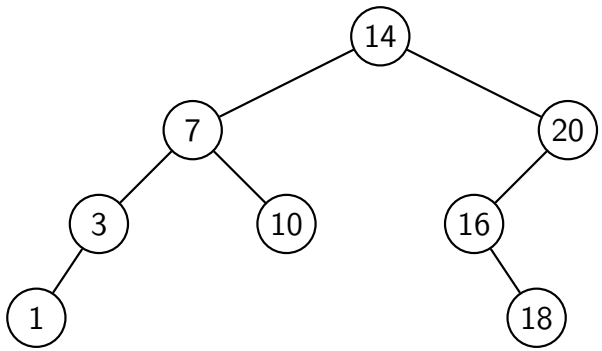
Type

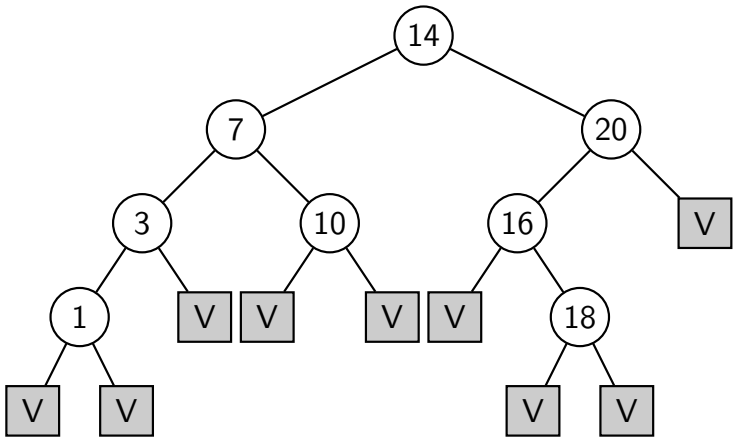
```
type 'a arbre = Vide | N of 'a * 'a arbre * 'a arbre ;;
```

Arbres binaires

Idée

arbre = vide (rien) ou étiquette + 2 sous-arbres





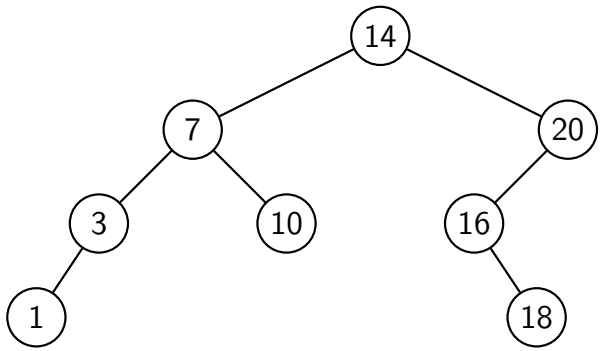
Type

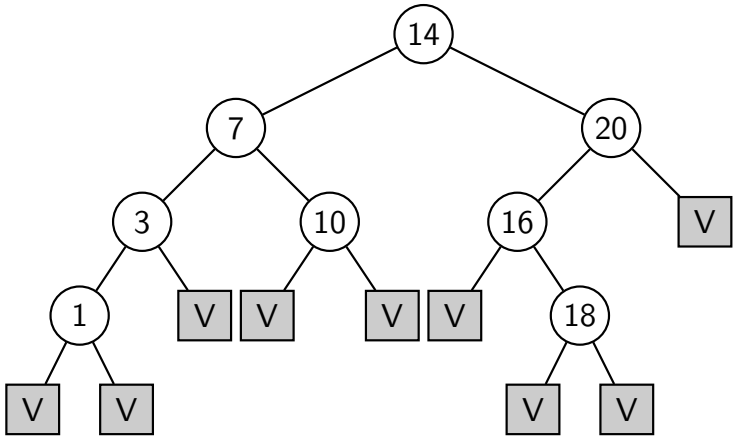
```
type 'a arbre = Vide | N of 'a * 'a arbre * 'a arbre ;;
```

Exemple

```
let ex_ab = N (14,
   N(7, N(3, N(1, Vide, Vide), Vide), N(10, Vide, Vide)),
   N(20, N(16, Vide, N(18, Vide, Vide)), Vide)) ;;
```

Exemple : hauteur





Hauteur

```
let rec hauteur a=match a with
  | Vide -> -1
  | N(_,g,d) -> 1 + max (hauteur g) (hauteur d)
;;
```