

CAML

8 - Arbres

<http://tsi.tuxfamily.org/OCaml>



25 avril 2023

Un premier exemple :

```
type couleur = Cyan | Magenta | Jaune | Melange
  of couleur * couleur ;;

let rouge = Melange (Magenta, Jaune);;
let orange = Melange (rouge , Jaune);;
```

Redéfinition d'une liste d'entiers :

```
# type liste_ent =  
    Vide  
    | Element of int * liste_ent;;  
  
# let l1 = Vide;;  
val l1 : liste_ent = Vide  
# let l2 = Element(2,Vide);;  
val l2 : liste_ent = Element (2, Vide)  
# let l3 = Element(5,l2);;  
val l3 : liste_ent = Element (5, Element (2,  
    Vide))
```

Redéfinition d'une liste d'entiers :

```
# type liste_ent =  
    Vide  
    | Element of int * liste_ent;;  
  
# let l1 = Vide;;  
val l1 : liste_ent = Vide  
# let l2 = Element(2,Vide);;  
val l2 : liste_ent = Element (2, Vide)  
# let l3 = Element(5,l2);;  
val l3 : liste_ent = Element (5, Element (2,  
    Vide))
```

On vient de définir la liste d'entiers [5,2].

Exercice

Définir sur le type `liste_ent` les fonctions :

- 1 tête
- 2 queue
- 3 longueur

```
let tete = function
  | Vide -> failwith "Liste vide"
  | Element (a, _) -> a;;
```

```
let tete = function
  | Vide -> failwith "Liste vide"
  | Element (a, _) -> a;;
```

```
let queue = function
  | Vide -> failwith "Liste vide"
  | Element (_, l) -> l;;
```

```
let tete = function
  | Vide -> failwith "Liste vide"
  | Element (a, _) -> a;;
```

```
let queue = function
  | Vide -> failwith "Liste vide"
  | Element (_, l) -> l;;
```

```
let rec longueur = function
  | Vide -> 0
  | Element (_, l) -> 1 + longueur l;;
```


Autre alternative : avec un constructeur infixe :

```
# type liste_ent =  
    Vide  
    | (::) of int * liste_ent;;  
type liste_ent = Vide | (::) of int * liste_ent  
  
# let l1 = Vide;;  
val l1 : liste_ent = Vide  
# let l2 = 2::Vide;;  
val l2 : liste_ent = (::) (2, Vide)  
# let l3 = 5::l2;;  
val l3 : liste_ent = (::) (5, (::) (2, Vide))
```

Dans l'exemple précédent, on aurait pu définir une liste d'objets quelconques ainsi :

```
#type 'a liste = Vide | Element of 'a * 'a liste  
;;
```

Le `type` `liste` est défini.

```
#let p1 = Vide;;
```

```
p1 : 'a liste = Vide
```

```
#let p2 = Element(2,Vide);;
```

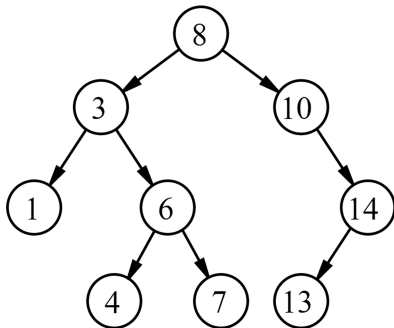
```
p2 : int liste = Element (2, Vide)
```

```
#let p3 = Element("Bob",Vide);;
```

```
p3 : string liste = Element ("Bob", Vide)
```

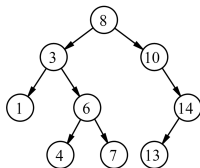
L'**arbre** est une structure de données qui généralise la liste : alors qu'une cellule de liste a un seul successeur, dans un arbre il peut y en avoir plusieurs. On parle alors de **nœud** (au lieu de cellule).

L'**arbre** est une structure de données qui généralise la liste : alors qu'une cellule de liste a un seul successeur, dans un arbre il peut y en avoir plusieurs. On parle alors de **nœud** (au lieu de cellule).



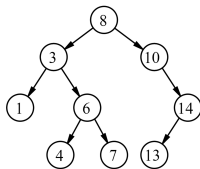
Vocabulaire

- Un **nœud père** peut avoir plusieurs **nœud fils**. Un fils n'a qu'un seul père.



Vocabulaire

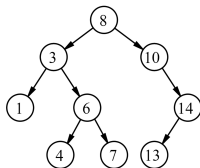
- Un **nœud père** peut avoir plusieurs **nœud fils**. Un fils n'a qu'un seul père.
- Tous les nœuds ont un ancêtre commun appelé la **racine de l'arbre** (le seul nœud qui n'a pas de père).



Racine : 8

Vocabulaire

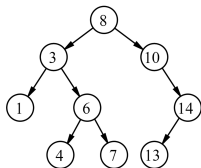
- Un **nœud père** peut avoir plusieurs **nœud fils**. Un fils n'a qu'un seul père.
- Tous les nœuds ont un ancêtre commun appelé la **racine de l'arbre** (le seul nœud qui n'a pas de père).
- Une **feuille** ou **nœud externe** est un nœud qui n'a pas de fils, sinon on parle de **nœud interne**.



Racine : 8
Feuilles : 1, 4, 7, 13

Vocabulaire

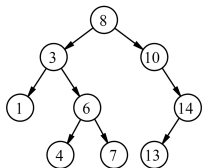
- Un **nœud père** peut avoir plusieurs **nœud fils**. Un fils n'a qu'un seul père.
- Tous les nœuds ont un ancêtre commun appelé la **racine de l'arbre** (le seul nœud qui n'a pas de père).
- Une **feuille** ou **nœud externe** est un nœud qui n'a pas de fils, sinon on parle de **nœud interne**.
- Le **degré** d'un nœud est le nombre de fils de ce nœud.



Racine : 8
Feuilles : 1, 4, 7, 13
6 est un nœud de degré 2.

Vocabulaire

- Un **nœud père** peut avoir plusieurs **nœud fils**. Un fils n'a qu'un seul père.
- Tous les nœuds ont un ancêtre commun appelé la **racine de l'arbre** (le seul nœud qui n'a pas de père).
- Une **feuille** ou **nœud externe** est un nœud qui n'a pas de fils, sinon on parle de **nœud interne**.
- Le **degré** d'un nœud est le nombre de fils de ce nœud.
- Un arbre dont chaque élément possède au plus deux fils est appelé **arbre binaire**



Un arbre binaire
Racine : 8
Feuilles : 1, 4, 7, 13
6 est un nœud de degré 2.

En OCaml, on peut définir un **arbre binaire** de cette manière :

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

En OCaml, on peut définir un **arbre binaire** de cette manière :

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

On accède ensuite aux éléments (aussi appelé **étiquette**) du type arbre par un filtrage de motifs du genre :

```
match t with  
  | Vide -> faire quelque chose  
  | Noeud (t1, x, t2) -> faire autre chose;;
```

Vocabulaire

On dit d'un arbre binaire qu'il est **strict** si tous ces nœuds internes ont une arité égale à 2.

Une autre méthode consiste alors à définir un arbre binaire strict non vide ainsi :

```
type ('a,'b) abs =  
  Feuille of 'a  
  | Noeud_int of 'b * ('a,'b) abs * ('a,'b) abs  
;;
```

Vocabulaire

- On appelle **profondeur** (ou niveau) d'un nœud, sa distance à la racine.
- La **hauteur** de l'arbre \mathcal{A} est la profondeur maximale de ses nœuds. Elle est définie inductivement par :
 - Si \mathcal{A} est vide, alors $h(\mathcal{A}) = -1$.
 - Sinon si \mathcal{A}_i sont les sous-arbres de \mathcal{A} ,
 $h(\mathcal{A}) = 1 + \max(h(\mathcal{A}_i))$.

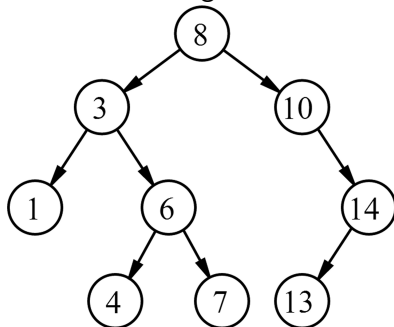
Vocabulaire

- On appelle **profondeur** (ou niveau) d'un nœud, sa distance à la racine.
- La **hauteur** de l'arbre \mathcal{A} est la profondeur maximale de ses nœuds. Elle est définie inductivement par :
 - Si \mathcal{A} est vide, alors $h(\mathcal{A}) = -1$.
 - Sinon si \mathcal{A}_i sont les sous-arbres de \mathcal{A} ,
 $h(\mathcal{A}) = 1 + \max(h(\mathcal{A}_i))$.

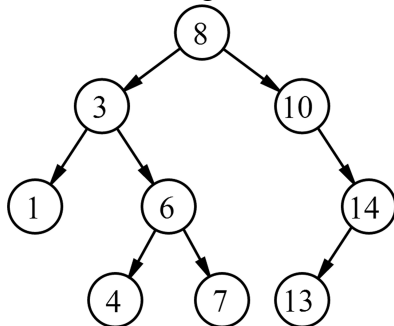
Propriété

Si un arbre binaire strict possède n nœuds (internes) et f feuilles alors $f = n + 1$

Préfixe gauche :

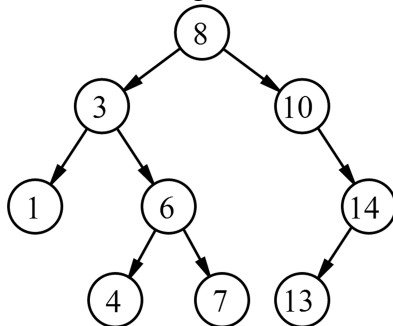


Préfixe gauche :

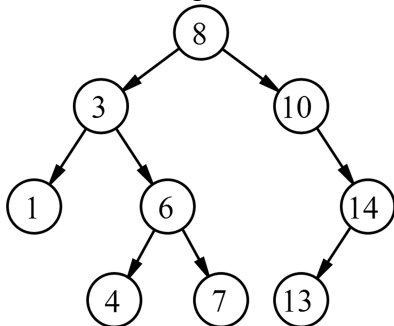


8 - 3 - 1 - 6 - 4 - 7 - 10 - 14 - 13

Infixe gauche :

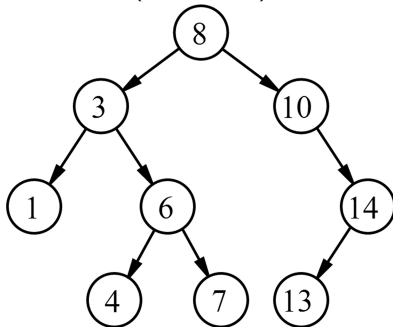


Infixe gauche :

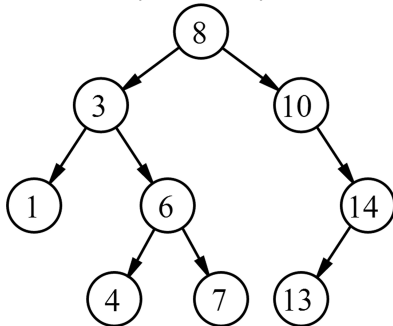


1 - 3 - 4 - 6 - 7 - 8 - 10 - 13 - 14

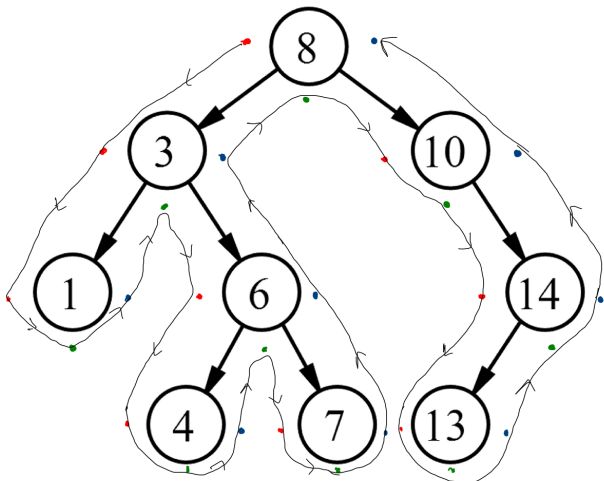
Postfixe (ou suffixe) gauche :



Postfixe (ou suffixe) gauche :

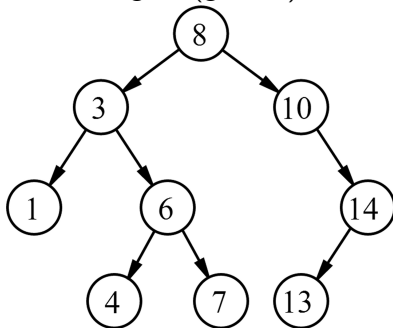


1 - 4 - 7 - 6 - 3 - 13 - 14 - 10 - 8

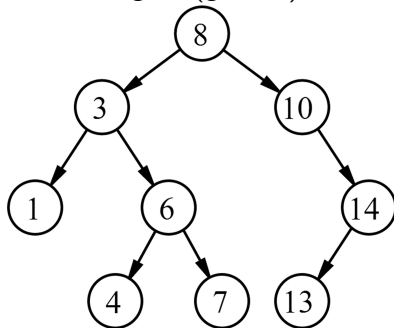


préfixe infixe suffixe

Largeur (gauche) :



Largeur (gauche) :



8 - 3 - 10 - 1 - 6 - 14 - 4 - 7 - 13

Arbre Binaire de Recherche

Un **arbre binaire de recherche** (ABR) est un arbre binaire dans lequel chaque nœud possède une valeur, telle que chaque nœud du sous-arbre gauche ait une valeur inférieure ou égale à celle du nœud considéré, et que chaque nœud du sous-arbre droit possède une valeur supérieure ou égale à celle-ci.

