Cours Informatique MP.

Arbres (généralités).

Antoine MOTEAU
antoine.moteau@wanadoo.fr

. . . /arbres.tex (2003)
. . . /arbres.tex Compilé le vendredi 09 mars 2018 à 10h 13m 06s avec LaTeX.
Compilable avec LaTeX, PDFLaTeX, LuaLaTeX, XeLaTeX.

Arbres (généralités)

Table des matières

Arbres (généralités)

1 Introduction

Définition 1.1.

Un arbre est un graphe orienté, tel que

(i) Il existe un unique sommet de degré entrant nul. Ce sommet est la **racine** de l'arbre.

(ii) Tout sommet distinct de la racine, est de degré entrant égal à 1.
L'origine y de l'unique arc arrivant sur x est appelée le **père** de x et x est dit **fils** de y.
Le nombre de fils d'un sommet y est **l'arité** de y .

(iii) Tout sommet est accessible à partir de la racine.

Remarques.

- Les sommets peuvent être affectés d'une valeur (ou **étiquette**) et les arcs (ou **branches**) ne sont pas étiquetés.
- Chaque sommet est la racine d'un sous-arbre, constitué des sommets accessibles depuis ce sommet et des arcs qui y conduisent.

1.1 Noeuds et feuilles

- les sommets internes (qui ont des fils) sont des **nœuds**.
- les sommets terminaux (qui n'ont pas de fils) sont des **feuilles**.

Il peut y avoir quelques variantes (et irrégularités) dans les dénominations . . . par exemple :

- lorsque les feuilles et les nœuds ont des natures semblables, tout sommet pourrait être qualifié de nœud, une feuille étant un **nœud terminal** (ou **nœud externe**) et un sommet interne étant un **nœud interne**.
- lorsque les feuilles et les nœuds ne sont pas de même nature, certains appellent **nœud terminal** un nœud qui n'a comme fils que des feuilles.

Si on veut parler de nœud ou de feuille, sans faire de distinction, on utilisera plutôt le mot **sommet**.

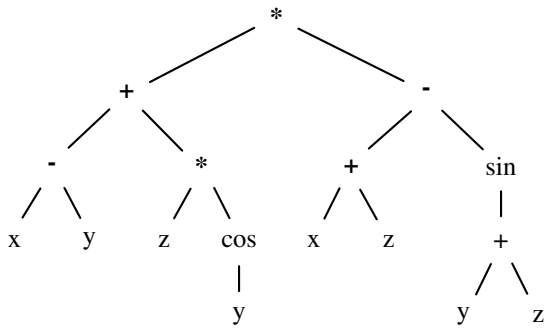
1.2 Etiquettes des sommets

Lorsque les nœuds sont munis d'étiquettes, celles ci sont prises dans un même ensemble de données.
Lorsque les feuilles sont munis d'étiquettes, celles ci sont prises dans un même ensemble de données qui peut être distinct de celui des étiquettes de nœuds.
Un arbre est dit

- **homogène** lorsque les étiquettes de feuilles et de nœuds sont de même nature ou lorsque les feuilles sont vides (sans étiquettes).
- **hétérogène** lorsque les étiquettes de feuilles et de nœuds ne sont pas de même nature.

1.3 Exemples

Arbres généalogiques
Arbres d'expressions algèbriques
Arbres binaires de recherche
Arbres de décision
etc . . .



$$((x-y)+(z*\cos(y)))*((x+z)-\sin(y+z))$$

1.4 Complément de vocabulaire spécifiques aux arbres

- L'**arité** (ou degré sortant) d'un nœud est le nombre de fils de ce nœud.
- Un arbre est d'arité k si tous les nœuds sont d'arité k.
- Un arbre binaire est un arbre d'arité 2, les deux fils de chaque nœuds étant, l'un le fils gauche, l'autre le fils droit.
- La **profondeur** d'un sommet est le nombre de branches (arcs) situées entre ce sommet et la racine.
- La **hauteur** d'un arbre est la plus grande profondeur atteinte pour une feuille.
- Si un sommet x est accessible depuis un sommet y, x est un **descendant** de y et y est un **ascendant** de x.
- On peut aussi introduire la notion de "grand-père", "petit-fils", "oncle", "neveu", "cousin", etc ...
- Une **forêt** est une "collection" (liste, par exemple) d'arbres de même nature.
- **...**

1.5 Variantes

On peut être amené à introduire certaines variantes à la notion d'arbre telle qu'elle a été présentée précédemment.

- Information complémentaire dans les sommets :
  - une couleur (pour la gestion de l'équilibre en hauteur entre fils droit et gauche),
  - une information sur l'équilibre de hauteur entre fils droit et gauche.
  - **...**
- Etiquettes composites, par exemple sous la forme de couples : (clé d'accès, données associées)
- Feuilles sans étiquettes ("vraies" feuilles ou "feuilles stériles") et feuilles avec étiquettes ("fausses" feuilles ou "fruits"), ce qui donne trois types de sommets :
  - les **nœuds**, dont le nœud racine, les nœuds internes, les nœuds terminaux,
  - les **fruits**,
  - les **feuilles**.
- Introduction d'un arc "de retour", d'un fils vers un père (c'est normalement non souhaitable)
- **...**

2 Type de données pour la représentation des arbres

L'ensemble des sommets accessibles depuis un sommet interne étant lui même un arbre, un type arbre se définit récursivement comme une structure de liens vers des sommets,

- chaque nœud interne contenant, outre son étiquette, des liens vers ses fils.
- les feuilles étant des arbres particuliers

En **Pascal**, les liens sont des pointeurs. Un arbre binaire simple, à étiquettes entières, se représente par :

```
Type
    ArbreBinaire = ^Noeud;                    { pointeur vers un objet de type Noeud }
    Noeud = record
                etiquette : integer;
                gauche, droit : ArbreBinaire
            end;
```

En **Caml**, la notion de lien est plus intégrée à la structure de donnée.

> En Caml, pour décrire le type de données, on utilise souvent un nom quasi-identique à celui d'un concept pour identifier la représentation de ce concept ou le moyen de créer (ou de décrire) cette représentation.
> La confusion de noms (par association bijective) permet une interprétation rapide des algorithmes.

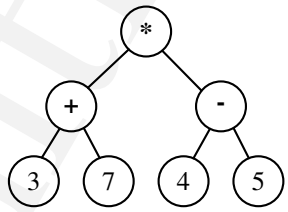
Dans la suite, on ne donne que des exemples en Caml.

2.1 Arbres binaires définis par union ou par enregistrement

2.1.1 Définition par union (fils gauche, étiquette père, fils droit)

On ne pourra **modifier un nœud qu'en créant un nouveau nœud** de remplacement.

```
type ('f ,'n) arbre =
  | Feuille of 'f                                    (* Feuille est un constructeur de lien *)
  | Noeud of (('f,'n) arbre) * 'n * (('f,'n) arbre)  (* Noeud   est un constructeur de lien *)
;;
let a = Noeud (     Noeud (Feuille 3, '+', Feuille 7),
                '*',
                    Noeud (Feuille 4, '-', Feuille 5) );;
(* ---> a : (int, char) arbre = Noeud (Noeud (Feuille 3, ... )) *)
```



- `Feuille` est le constructeur de lien vers une feuille de type `'f`
- `Noeud` est le constructeur de lien vers un nœud de type `(('f,'n) arbre) * 'n * (('f,'n) arbre)`

On utilise aussi les constructeurs comme des descripteurs dans les fonctions de sélection. Par exemple :

```
let max_valeur t = aux 0 t        (* maximum des étiquettes de feuilles *)
   where rec aux accu = function
     | Feuille f    -> max accu f                (* Feuille utilisé comme descripteur *)
     | Noeud(g,e,d) -> aux (aux accu g) d         (* Noeud   utilisé comme descripteur *)
;;
(* max_valeur : (int, 'a) arbre -> int = <fun> *)
max_valeur a;;  (* ---> 7 *)
```

2.1.2 Définition sous forme d'enregistrement, à champs mutables

Pour pouvoir **modifier un nœud sans en créer de nouveau**, on représente les données par des enregistrements, avec des champs modifiables en place (mutables). La gestion de ces arbres est un peu délicate.

```
type Noeud =
   {mutable etiquette : int; mutable gauche : ArbreBinaire; mutable droit : ArbreBinaire}
and ArbreBinaire =    (* structure arborescente de "liens" *)
        ArbreVide            (* ArbreVide est le constructeur de "lien vide" ou "feuille" *)
      | Pointeur of Noeud    (* Pointeur est le constructeur de lien                      *)
                             (*     vers un "noeud" qui est de type Noeud                 *)
;;
let noeud = function                    (* noeud est une fonction de sélection *)
   | ArbreVide  -> failwith "arbre vide"          (* constructeur utilisé comme descripteur *)
   | Pointeur x -> x                              (* constructeur utilisé comme descripteur *)
;;
(* noeud : ArbreBinaire -> Noeud = <fun> *)

let a = Pointeur {etiquette = 1;
                  gauche = Pointeur {etiquette = 2; gauche = ArbreVide; droit = ArbreVide};
                  droit = ArbreVide};;
(* ---> a : ArbreBinaire = Pointeur {etiquette = 2; ...  } *)
```

2.1.3 Définition sous forme d'enregistrement, avec lien au père (à éviter)

Chaque nœud étant capable d'accéder à son père, on peut se déplacer indifféremment de la racine vers les feuilles ou des feuilles vers la racine. La gestion de tels arbres pourrait s'avérer particulièrement délicate ...

```
type Noeud =
   {mutable etiquette: int; mutable gauche: ArbreBinaire; mutable droit: ArbreBinaire;
    mutable pere : ArbreBinaire }
and ArbreBinaire =
        ArbreVide            (* ArbreVide est le constructeur de "lien vide" (ou "feuille") *)
      | Pointeur of Noeud    (* Pointeur est le constructeur de lien vers un "noeud"        *)
;;
```

2.2 Exemples d'arbres binaires définis par union

2.2.1 Arbre binaire homogène à feuilles vides

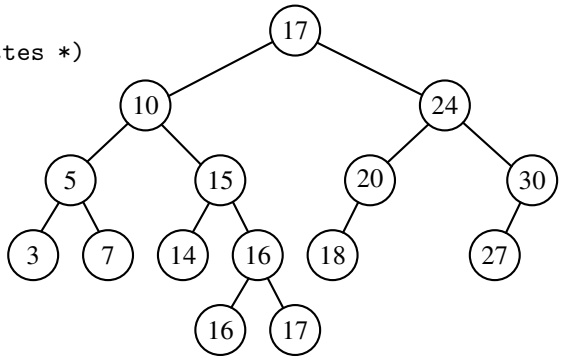
```
type 'a arbre =
  | Vide                                (* Vide est le constructeur de "lien vide" ou "feuille" *)
  | N of ('a arbre) * 'a * ('a arbre)   (* N est le constructeur de lien vers un "noeud" *)
;;
```

On pourra utiliser ce type pour représenter un Arbre Binaire de Recherche :

```
let rec insert_abr k = function   (* insertion avec reconstruction complète d'un arbre nouveau *)
  | Vide     -> N(Vide,k,Vide)
  | N(g,e,d) -> if k <= e then N(insert_abr k g , e , d)
                          else N(g , e , insert_abr k d)
;;
(* insert_abr : 'a -> 'a arbre -> 'a arbre = <fun> *)

let list_of_abr = aux []                (* exemple de parcours en profondeur "gpd" *)
   where rec aux accu = function
        | Vide -> accu
        | N(g,e,d) -> aux (e::aux accu d) g
;;
(* list_of_abr : '_a arbre -> '_a list = <fun> *)                    (* Type faible !!!! *)

let a = list_it (insert_abr) [16;17;27;18;16;7;30;14;15;20;3;24;5;10;17] Vide;;
list_of_abr a;;  (* [1; 3; 5; 5; 7; 10; 12; 14; 16; 17; 18; 20; 24; 27; 30] *)
```

Exemple de calculs :

```
let rec somme = function   (* somme des étiquettes *)
    | Vide -> 0
    | N(g,e,d) -> e + (somme g) + (somme d)
;;
(* somme : int arbre -> int = <fun> *)
somme a;;  (* ---> int = 239 *)
```



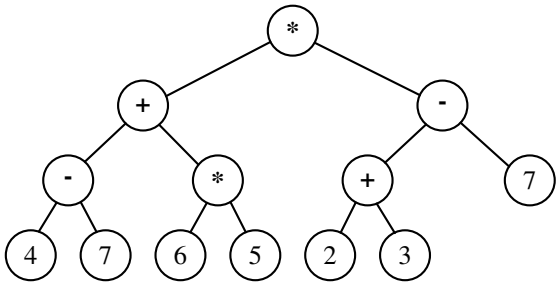
2.2.2 Arbre binaire hétérogène à deux types d'étiquettes

```
type ('f ,'n) arbre =
  | F of 'f
  | Noeud of (('f,'n) arbre) * 'n * (('f,'n) arbre)
;;
```

On peut utiliser ce type d'arbre pour représenter des expressions arithmétiques (simples) :

```
let a = Noeud (Noeud (Noeud (F 4, '-', F 7), '+', Noeud (F 6, '*', F 5)), '*',
               Noeud (Noeud (F 2, '+', F 3), '-', F 7))
;;
```

```
let rec eval = function
  | F x -> x
  | Noeud(g,e,d) when e = '+' -> (eval g) + (eval d)
  | Noeud(g,e,d) when e = '-' -> (eval g) - (eval d)
  | Noeud(g,e,d) when e = '*' -> (eval g) * (eval d)
  | _ -> failwith "opérateur non reconnu"
;;
(* eval : (int, char) arbre -> int = <fun>*)

eval a;;        (* #- : int = -54 *)
```



2.2.3 Arbre binaire "hétérogène" à un seul type d'étiquettes

```
type 'a arbre =
  | Feuille of 'a
  | Noeud of ('a arbre) * 'a * ('a arbre)  (* (fils gauche, étiquette père, fils droit) *)
;;

let rec somme = function    (* somme des étiquettes *)
    | Feuille x -> x
    | Noeud (fg, ep, fd) -> ep + (somme fg) + (somme fd)
;;
(* somme : int arbre -> int = <fun> *)
```

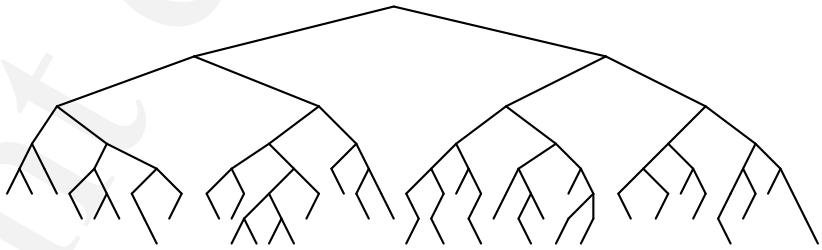
La fonction somme, qui ne s'applique qu'au type `int arbre`, est un exemple de parcours en profondeur.

```
let a = Noeud( Noeud(Feuille 3, 5, Feuille 7), 8, Feuille 10 );;
somme a;;       (* --> 33 *)
```

Remarques. Ce type d'arbre

- peut être plus simplement remplacé par le type "arbre homogène" à feuilles vides vu ci-dessus.
- peut être un cas particulier du type "arbre hétérogène" à deux sortes d'étiquettes vu ci-dessus.

2.2.4 Arbre binaire squelettique



```
type arbre =
  | F
  | N of arbre * arbre
;;

let rec hauteur = function
    | F -> 0
    | N (g, d) -> 1 + max (hauteur g) (hauteur d)
;;
(* hauteur : arbre -> int = <fun> *)

hauteur ( N(N(F,F),N(F,N(F,N(N(F,N(N(F,F),F)),F)))) );;  (* --> 7 *)
```

Un arbre squelettique peut être utilisé pour décrire des possibilités de cheminement, sans être distrait par des données éventuelles (réseau routier, sans les étapes gastronomiques . . .).

2.3 Exemples d'arbres d'arité variable

2.3.1 Arbre hétérogène d'arité variable

On pourrait envisager une telle structure pour représenter l'arborescence des dossiers stockés sur un disque.

```
type ('f, 'n) arbre =
  | F of 'f                              (* fichier *)
  | N of 'n * (('f,'n) arbre list)       (* dossier de fichiers (ou répertoire) *)
;;

let max_nb_fils = aux 0
  where rec aux k = function
    | F x -> k
    | N (e, u) -> max (list_length u) (it_list aux k  u)
;;
(* max_nb_fils : ('_a, '_b) arbre -> int = <fun> *)                    (* type faible *)

let a = N( 1, [N(2,[F 'a'; F 'b'; F 'b']); N(3, [])]);;
max_nb_fils a;;         (* --> 3 *)
```

Le premier appel à `max_nb_fils` en a fixé le type :

```
max_nb_fils;;  (* ---> - : (char, int) arbre -> int = <fun> *)
```

2.3.2 Arbre homogène, à un seul type de sommets, d'arité variable

```
type 'a arbre = Noeud of 'a * ('a arbre list)
;;
```

(un nœud terminal, à liste de fils vide, joue ici le rôle de feuille étiquetée)

2.3.3 Forêt d'Arbres

```
type 'n arbre =
  | Feuille of 'n                          (* Feuille est un constructeur  *)
  | N of 'n * ('n foret)                   (* N est un constructeur        *)
and 'n foret = Arbres of 'n arbre list     (* Arbres est un constructeur   *)
;;
```

Recherche du nombre maximal de fils réalisé dans un arbre :

```
let max_nb_fils t = aux 0 t              (* si t est omis, on a un type faible *)
  where rec aux k = function
    | Feuille x -> k
    | N (e, Arbres u) -> it_list aux (max k (list_length u)) u
;;
(* max_nb_fils : 'a arbre -> int = <fun> *)                          (* type fort *)

let a = N( 1, Arbres [N(2, Arbres [Feuille 7; Feuille 4; Feuille 5]); N(5, Arbres [])]);;
max_nb_fils a;;         (* --> 3 *)
```

2.3.4 Arbre d'arité k (variable), défini à l'aide de vecteurs

```
type ('f,'n) arbre =
  | F of 'f
  | Noeud of 'n * (('f,'n) arbre vect)
;;

let rec is_arite_fixe k = function
    | F x -> true
    | Noeud (e, v) -> (vect_length v = k) &
                      (try for i = 0 to k-1
                           do if not (is_arite_fixe k v.(i)) then raise Exit done;
                           true
                       with Exit -> false)
;;
(* is_arite_fixe : int -> ('a, 'b) arbre -> bool = <fun> *)

let b = Noeud( 1, [| Noeud(2, [|F `a`; F `b`|]); Noeud(3, [|F `c`; F `d`; F `e`|]) |]);;
is_arite_fixe 2 b;;          (* --> bool = false *)
```

2.3.5 ...

3 Théorèmes élémentaires

- Lorsque les arbres sont définis inductivement, on a naturellement des **démonstrations par induction**, les feuilles représentant le cas de base et les nœuds le cas général.
- Le nombre de sommets d'un arbre fils étant strictement inférieur à celui de l'arbre père, on a naturellement des **démonstrations par récurrence** sur le nombre de sommets.
- La hauteur d'un arbre fils étant strictement inférieur à celle de l'arbre père, on a naturellement des **démonstrations par récurrence** sur la hauteur.

3.1 Branches et feuilles

Théorème 3.1.

Un arbre à n sommets possède exactement n-1 branches (arêtes).

Preuve. Par induction structurelle (ou récurrence sur le nombre de sommets) :

- Si l'arbre est réduit à une feuille, il a 1 sommet et 0 arêtes.
- Sinon, soit A un arbre à $n > 1$ sommets et r sa racine. Les p fils $A_1, A_2, \cdots, A_p$ de la racine r ont respectivement $n_1, n_2, \cdots, n_p$ sommets (avec $n = 1 + n_1 + n_2 + \cdots + n_p$).
  Par hypothèse d'induction, chaque arbre A_k possède $n_k - 1$ arêtes et comme il y a p arêtes reliant r à ses p fils, on obtient en tout $p + (n_1 - 1) + (n_2 - 1) + \cdots + (n_p - 1) = n_1 + n_2 + \cdots + n_p = n - 1$ arêtes pour l'arbre A, ce qui termine la preuve par induction (ou par récurrence)

Théorème 3.2.

Un arbre binaire à n nœuds possède n+1 feuilles.

Preuve. Par induction structurelle (ou récurrence sur le nombre de nœuds) :

- Si l'arbre est réduit à une feuille, il a 0 nœud et 1 feuille.
- Sinon, soit A un arbre à $n > 1$ nœuds et r sa racine. Les 2 fils A_1 et A_2 de la racine r ont respectivement n_1 et n_2 nœuds (avec $n = 1 + n_1 + n_2$).
  Par hypothèse d'induction, A_1 et A_2 ont $n_1 + 1$ et $n_2 + 1$ feuilles et on en déduit que A possède $n_1 + 1 + n_2 + 1 = n + 1$ feuilles, ce qui termine la preuve par induction (ou par récurrence)

Plus généralement, dans un arbre d'arité fixe k, s'il y a n nœuds alors il y a $(k-1)n + 1$ feuilles.

3.2 Encadrement de la hauteur

Théorème 3.3.

Dans un arbre dont les n nœuds sont tous de degré inférieur ou égal à 2, la hauteur h vérifie :

$$h \leqslant n \leqslant 2^h - 1 \qquad \left(\textit{ou encore } \log_2(n+1) \leqslant h \leqslant n \right)$$

Preuve. Par induction structurelle (ou récurrence sur le nombre de nœuds) :

- Si l'arbre est réduit à une feuille, il a $n = 0$ nœuds et 1 feuille. On a alors $h = n = 0$ et la double inégalité est vérifiée.
- Sinon, soit A un arbre à $n > 1$ nœuds, de hauteur h et de racine r.
  - Si r n'a qu'un fils U, ce fils U possède $n - 1$ nœuds et a pour hauteur $h - 1$. Par induction structurelle, on a $h - 1 \leqslant n - 1 \leqslant 2^{h-1} - 1$ et on en déduit $h \leqslant n \leqslant 2^{h-1}$ et on a $2^{h-1} \leqslant 2^h - 1$
  - Si r a deux fils, U et V, ces fils sont de hauteurs respectives u et v et ont respectivement a et b nœuds. Par induction structurelle, on a $u \leqslant a \leqslant 2^u - 1$ et $v \leqslant b \leqslant 2^v - 1$.
    Comme $h = 1 + \max(u, v)$ on en déduit :
    - $n = 1 + a + b \geqslant 1 + u + v \geqslant 1 + \max(u, v) = h$
    - $n = 1 + a + b \leqslant 2^u + 2^v - 1 \leqslant 2 \times 2^{\max(u,v)} - 1 = 2^{1+\max(u,v)} - 1 = 2^h - 1$

  Ce qui termine la preuve par induction (ou par récurrence)

Plus généralement, dans un arbre à n noeuds d'arité au plus k, la hauteur h vérifie $h \leqslant n \leqslant \dfrac{k^h - 1}{k - 1}$.

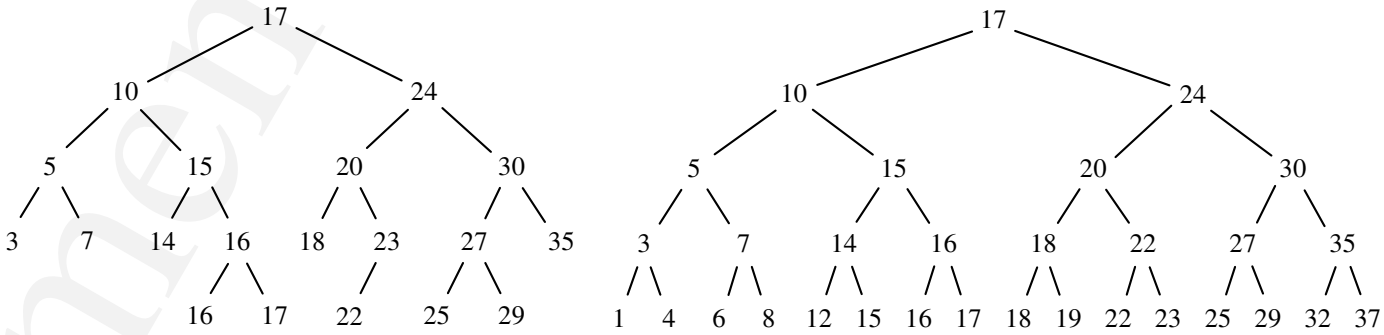
3.3 Arbres binaires équilibrés

3.3.1 Equilibre strict

Définition 3.1. Arbre binaire strictement équilibré, arbre binaire complet :

- Un arbre binaire de hauteur h est dit **strictement équilibré** (totalement équilibré) si tous les chemins menant de la racine à une feuille ont pour longueur h ou h-1.
- Si tous les chemins menant de la racine à une feuille ont pour longueur h, l'arbre est dit **complet** (tous les niveaux sont remplis).

Remarque. Ces notions pourraient s'étendent aux arbres k-aires.



Théorème 3.4. *Hauteur d'un arbre binaire strictement équilibré.*

Dans un arbre binaire strictement équilibré, de hauteur h, à n nœuds (et $p = n+1$ feuilles), on a :

$$2^{h-1} \leqslant n \leqslant 2^h - 1 \qquad \left(\textit{ ou encore } \log_2(n+1) \leqslant h \leqslant 1 + \log_2 n \right)$$

De plus, si l'arbre est complet alors $n = 2^h - 1$.

Preuve. La propriété est vraie pour $n = 0$ et on la démontre pour $n > 0$:

(il ne s'agit pas d'une démonstration par récurrence).
On sait déjà que $n \leqslant 2^h - 1$ et il ne reste plus qu'à montrer que $2^{h-1} \leqslant n$, par exemple en comptant les nœuds qui sont situés à une distance d (pour d allant de 0 à h-1) de la racine :

- il y en a 1 à la distance 0, 2 à la distance 1 (si $1 < h-1$) et, par récurrence (simple) , il y en a $2k$ à la distance k lorsque $0 \leqslant k < h-1$.
- il y a au moins un nœud à la distance $h-1$

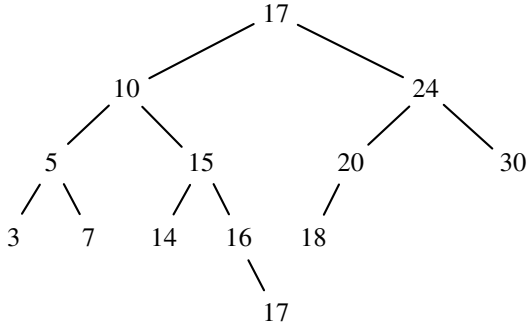
on en déduit que $n \geqslant 1 + \cdots 2^{h-2} + 1 = (2^{h-1} - 1) + 1 = 2^{h-1}$

Conséquence : Dans un arbre binaire strictement équilibré à n nœuds internes, de hauteur $h = h(n)$, on a

$$h(n) \underset{+\infty}{\sim} \log_2(n) \qquad (\text{ plus précis que } h(n) = O(\ln n))$$

3.3.2 H-Equilibre

A défaut d'avoir un équilibre strict, coûteux à maintenir lors des insertions ou suppressions de nœuds, on peut se contenter d'une notion d'équilibre plus faible, moins coûteuse à maintenir, tout en conservant une hauteur qui est encore un $O(\ln n)$.



Définition 3.2. Arbre binaire H-équilibré :

Un arbre (binaire) est dit **H-équilibré** (partiellement équilibré) si pour tout nœud, la différence de hauteur entre les arbres fils gauche et fils droit est au plus un.

Théorème 3.5. *Hauteur d'un arbre binaire H-équilibré.*

Dans un arbre binaire H-équilibré, de hauteur h, à n nœuds on a :

$$h \leqslant 2.08\ln(n+2) - 0.32 \qquad \left(\textit{ou } h \leqslant 1.45\log_2(n+2) - 0.32 \right)$$

ce qui est plus précis que $h = O(\ln n)$.

Preuve. On construit pour chaque hauteur h, un arbre H-équilibré de hauteur h comportant un nombre minimal $N(h)$ de sommets :

- pour $h = 0$, on a $N(h) = 0$
- pour $h = 1$, on a $N(h) = 1$
- si $h > 1$, pour que le nombre de sommet soit minimal, un des fils doit être de hauteur $h-1$ et l'autre fils doit être de hauteur $h-2$. On aura donc $N(h) = N(h-1) + N(h-2) + 1$

En posant $u_h = N(h) + 1$, on a $\begin{cases} u_0 = 1 \quad ; \quad u_1 = 2 \\ u_{h+2} = u_{h+1} + u_h \end{cases}$ (récurrence linéaire à coefficients constants) et on obtient $u_h = \dfrac{1}{\sqrt{5}}\left(\dfrac{1+\sqrt{5}}{2}\right)^{h+2} - \dfrac{1}{\sqrt{5}}\left(\dfrac{1-\sqrt{5}}{2}\right)^{h+2} \underset{h\to+\infty}{\sim} \dfrac{1}{\sqrt{5}}\left(\dfrac{1+\sqrt{5}}{2}\right)^{h+2}$.

On a aussi $u_h \geqslant \dfrac{1}{\sqrt{5}}\left(\dfrac{1+\sqrt{5}}{2}\right)^{h+2} - 1$ et $\dfrac{1}{\ln\left(\dfrac{1+\sqrt{5}}{2}\right)} \approx 2.078086921.$

On a donc $N(h) + 2 \geqslant \dfrac{1}{\sqrt{5}}\left(\dfrac{1+\sqrt{5}}{2}\right)^{h+2}$ d'où $h \leqslant \log_{\frac{1+\sqrt{5}}{2}}\left((n+2)\sqrt{5}\right) - 2$, soit

$$h \leqslant \log_{\frac{1+\sqrt{5}}{2}}(n+2) + \log_{\frac{1+\sqrt{5}}{2}}\left(\sqrt{5}\right) - 2$$

ce qui, par valeurs approchées, avec majoration, donne $h \leqslant 2.08\ln\big(N(h)+2\big) - 0.32$.

3.3.3 Autres notions d'équilibre

On peut introduire d'autres notions d'équilibre ou de quasi équilibre, toutes ayant pour objectifs

- d'obtenir une hauteur qui soit un $O\big(\ln n\big)$,
- de maintenir l'équilibre avec un coût faible lors des insertions ou suppressions de nœuds.

Ces notions d'équilibres, associées à des organisations particulières des données contenues dans les étiquettes et aux méthodes de maintien de l'équilibre lors des opérations d'insertions et de suppressions, conduisent à des familles d'arbres (structures et méthodes) particulières :

- Arbres AVL (Adel'son, Vel'skii, Landis, 1962).
  L'équilibre est maintenu par rotations et $O(\ln n)$ rotations peuvent être nécessaires pour maintenir l'équilibre après une suppression.
- Arbres 2-3 (Hopcroft 1970). L'équilibre est maintenu en manipulant le degré des nœuds.
- B-Arbres (Bayer, McCreight, 1972) généralisation des arbres 2-3 .
- Arbres Rouge et Noir (Bayer, Sleator, Tarjan), (et aussi Guibas, Sedgewick).
- Arbres déployés (Sleator, Tarjan, 1983).
- ...

4 Parcours d'un arbre

Comme dans le cas des graphes, on a essentiellement deux types de parcours des arbres :

- Le **parcours en profondeur**, où, depuis la racine, pour chaque nœud, chaque arbre fils est visité entièrement avant que les autres ne le soient.
- Le **parcours en largeur**, où, depuis la racine, tous les nœuds situés à la distance n de la racine sont visités avant tous les nœuds situés à la distance n+1

On peut être amené à limiter les parcours :

- lorsque l'on ne peut pas parcourir toute la structure de l'arbre en un temps fini, on indiquera une profondeur maximale de parcours et on se contentera de résultats partiels (choisir une "meilleure" stratégie parmi celles dont on peut estimer la pertinence en un temps fini de calcul).
- une évaluation pourrait conduire à l'abandon de l'exploration d'un sous arbre (impasse prévisible ou solution moins bonne qu'une solution précédemment évaluée)

4.1 Parcours en profondeur

C'est le type de parcours le plus aisé à réaliser et il est en général accompagné d'un "traitement" dit

- **préfixe** lorsque le père est traité avant ses fils,
- **postfixe** lorsque le père est traité après ses fils
- **infixe** (pour les arbres binaires) lorsque le père est traité entre ses deux fils

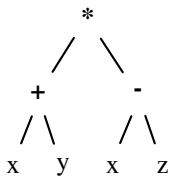
Pour les exemples, on se place dans le cadre restreint des arbres binaires hétérogènes :

```
type ('f ,'n) arbre =
  | F of 'f                                    (* F, Vide, Nil, Feuille, Fruit, ... *)
  | N of ('f,'n) arbre) * 'n * (('f,'n) arbre)    (* N, Noeud, ...                    *)
;;
```

4.1.1 Exemples élémentaires de parcours en profondeur

- On a déjà vu, dans les exemples précédents, quelques parcours élémentaires en profondeur.
- Expression $\left\{ \begin{array}{l} \text{algébrique totalement parenthésée (EATP)} \\ \text{polonaise inverse (RPN)} \end{array} \right\}$ issue d'un arbre d'expression arithmétique :

```
let rec print_alg = function
  | F x      -> print_char x
  | N(g,e,d) -> print_char '('; print_alg g; print_char e; print_alg d; print_char ')';
;;
(* print_alg : (char, char) arbre -> unit = <fun> *)

let rec print_rpn = function
  | F x      -> print_char x
  | N(g,e,d) -> print_rpn g; print_char ','; print_rpn d; print_char ',';print_char e;
;;
(* print_rpn : (char, char) arbre -> unit = <fun> *)

let a = N( N(F 'x', '+', F 'y'), '*', N(F 'x', '-', F 'z') );;
print_alg a;;   (* --> ((x+y)*(x-z))  *)
print_rpn a;;   (* --> x,y,+,x,z,-,*  *)
```



4.1.2 Prototypes de parcours en profondeur

En s'inspirant des fonctionnelles `do_list`, `it-list`, `map`, on peut introduire des modèles de parcours, admettant en paramètres des informations de traitement particulières :

- Parcours avec effets annexes (modèle `do_xxx_arbre`)
- Parcours avec évaluations accumulées (modèle `it_xxx_arbre`)
- Parcours avec transformations (modèle `map`)

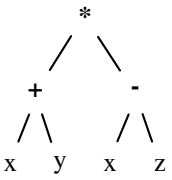
Dans le nom des prototypes, l'indication **gdp** , **gpd** , etc ... indique l'ordre dans lequel se réalise le traitement du fils gauche (**g**), du père (**p**) et du fils droit (**d**), lorsque cet ordre a un sens.

```
let rec do_gdp_arbre ff fn = function                          (* gdp == postfixe *)
  | F x      -> ff x
  | N(g,e,d) -> do_gdp_arbre ff fn g; do_gdp_arbre ff fn d; fn e
;;
(* do_gdp_arbre : ('a -> 'b) -> ('c -> 'b) -> ('a, 'c) arbre -> 'b = <fun> *)

let rec it_gpd_arbre ff fn vf vn = function                    (* gpd == infixe *)
  | F x      -> (ff vf x, vn)
  | N(g,e,d) -> let a,b = it_gpd_arbre ff fn vf vn g in it_gpd_arbre ff fn a (fn b e) d
;;
(* it_gpd_arbre : ('a -> 'b -> 'a) ->
         ('c -> 'd -> 'c) -> 'a -> 'c -> ('b, 'd) arbre -> 'a * 'c = <fun> *)

let rec map_arbre ff fn = function
  | F x      -> F (ff x)
  | N(g,e,d) -> N(map_arbre ff fn g, fn e, map_arbre ff fn d)
;;
(* map_arbre : ('a -> 'b) -> ('c -> 'd) -> ('a, 'c) arbre -> ('b, 'd) arbre = <fun> *)

let rec hom_arbre fn ff = function    (* homomorphisme *)
  | F x      -> ff x
  | N(g,e,d) -> fn (hom_arbre fn ff g, e, hom_arbre fn ff d)
;;
(* hom_arbre : ('a * 'b * 'a -> 'a) -> ('c -> 'a) -> ('c, 'b) arbre -> 'a = <fun> *)
```

Exemples de mise en œuvre des prototypes :

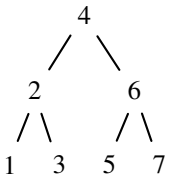
- Expression polonaise inverse d'une expression arithmétique

```
let print_rpn = do_gdp_arbre f f
  where f x = print_char x; print_char ' '
;;
(* print_rpn : (char, char) arbre -> unit = <fun> *)

let a = N( N(F 'x', '+', F 'y'), '*', N(F 'x', '-', F 'z') );;
print_rpn a;;  (* --> x y + x z - *  *)
```



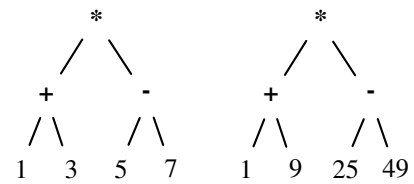
- Somme des étiquettes de feuilles et produit de celles des noeuds

```
let a = N( N(F 1, 2, F 3), 4, N(F 5, 6, F 7) );;
it_gpd_arbre (prefix +) (prefix *) 0 1 a;;  (* --> 16, 48 *)
```



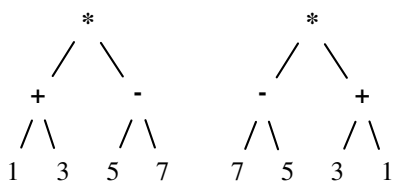
- Transformation en un arbre où les étiquettes de feuilles sont élevées au carré

```
let a = N( N(F 1, '+', F 3), '*', N(F 5, '-', F 7) );;
map_arbre (function x -> x*x) (function x -> x) a;;
(* --> (int,char) arbre
  = N(N(F 1,'+',F 9),'*',N(F 25,'-',F 49)) *)
```



- Miroir d'un arbre par homomorphisme

```
let miroir = hom_arbre (fun (g,e,d) -> N(d,e,g)) (fun x -> F x)
;;
(* miroir : ('_a, '_b) arbre -> ('_a, '_b) arbre = <fun> *)      (* type faible *)

let a = N( N(F 1, '+', F 3), '*', N(F 5, '-', F 7) );;
miroir a;;
(* (int,char) arbre=N(N(F 7,'-',F 5),'*',N(F 3,'+',F 1)) *)
```



Remarque : Le type "faible" (provisoire) a été fixé (définitivement) par le premier appel.

```
miroir;; (* (int, char) arbre -> (int, char) arbre = <fun> *)
```

4.2 Parcours en largeur

Pour réaliser un tel parcours, il est naturel d'utiliser une **file FIFO** (First In First Out) dont le contenu est initialement réduit à la racine de l'arbre.

4.2.1 Algorithme

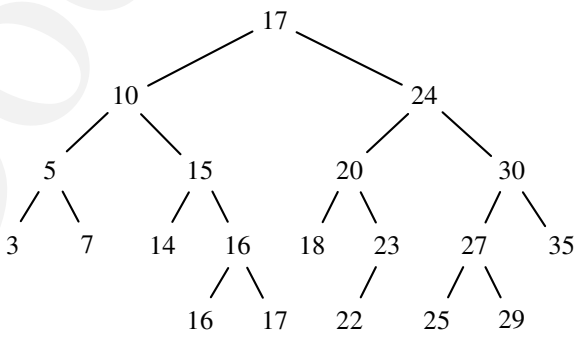
- La racine est introduite dans une file FIFO neuve
- Tant que la file n'est pas vide, on extrait l'élément de tête de la pile pour
  - appliquer une fonction à son étiquette (traitement réalisé par le parcours),
  - introduire ses fils éventuels dans la pile.

4.2.2 Exemple

```
#open "queue";;  (* module Caml : queues (FIFOs), modifiables sur place.*)

type ('f ,'n) arbre =
  | F of 'f
  | N of (('f,'n) arbre) * 'n * (('f,'n) arbre)
;;

let parcours_large ff fn a =
  let Fifo = queue__new () in queue__add a Fifo;

  while queue__length Fifo > 0
  do traite (queue__take Fifo) done

  where traite = function
    | F x      -> ff x
    | N(g,e,d) -> queue__add g Fifo; queue__add d Fifo; fn e
;;
(* parcours_large : ('a -> 'b) -> ('c -> 'b) -> ('a, 'c) arbre -> unit = <fun> *)
```

On peut en déduire l'édition du contenu d'un arbre, couche après couche :

```
let print_large t = parcours_large ff fn t
    where ff x = () and fn x = print_int x; print_char ' '
;;
(* print_large : ('a, int) arbre -> unit = <fun> *)

let a = N (N (N (N (F 'f', 3, F 'f'), 5, N (F 'f', 7, F 'f')), 10,
     N (N (F 'f', 14, F 'f'), 15, N (N (F 'f', 16, F 'f'), 16, N (F 'f', 17, F 'f')))), 17,
   N(N (N (F 'f', 18, F 'f'), 20, N (N (F 'f', 22, F 'f'), 23, F 'f')), 24,
     N (N (N (F 'f', 25, F 'f'), 27, N (F 'f', 29, F 'f')), 30, N (F 'f', 35, F 'f'))))
;;
```




```
print_large a;;  (* --> 17 10 24 5 15 20 30 3 7 14 16 18 23 27 35 16 17 22 25 29 *)
```

5 Longueur moyenne des chemins de la racine à un sommet, dans un arbre binaire.

Théorème 5.1.

Dans un arbre binaire à n sommets, la longueur moyenne $L_m(n)$, des chemins allant de la racine à un sommet, vérifie

$$L_m(n) = O(\ln n) \qquad \left(\text{et même } L_m(n) \underset{n\to+\infty}{\sim} 2\ln n\right)$$

Preuve. On note L à la place de L_m ... .
On a $L(0) = 0$ (et $L(1) = 0$) et, pour $n \geqslant 1$, avec un arbre à n sommets :

- Lorsque l'arbre gauche A_g, issu de la racine, possède k sommets, l'arbre droit A_d , issu de la racine, possède $n-k-1$ sommets et la probabilité pour qu'un sommet quelconque x soit

dans A_g est	$\frac{k}{n}$	avec une longueur d'accès moyen	$L(k)+1$
dans A_d est	$\frac{n-k-1}{n}$	avec une longueur d'accès moyen	$L(n-k-1)+1$
la racine est	$\frac{1}{n}$	avec une longueur d'accès	0

  La longueur d'accès moyen à x est donc $\dfrac{k\left(L(k)+1\right)+(n-k-1)\left(L(n-k-1)+1\right)+0}{n}$

- Pour un arbre à n sommets, construit aléatoirement, toutes les valeurs de k entre 0 et $n-1$ sont équiprobables. La longueur d'accès moyen à un sommet quelconque est donc

$$L(n) = \frac{1}{n}\sum_{k=0}^{n-1} \frac{k\left(L(k)+1\right)+(n-k-1)\left(L(n-k-1)+1\right)}{n}$$
$$= \frac{n-1}{n} + \frac{2}{n^2}\sum_{k=0}^{n-1} kL(k) = \frac{n-1}{n} + \frac{2}{n^2}\sum_{k=1}^{n-1} kL(k)$$

1. On montre par récurrence, sur $n > 0$, que $L(n) \leqslant 4\ln n$ (et donc que $L(n) = O(\ln n)$) :
   - c'est vrai pour $n = 1$: $L(1) = 0 \leqslant 4\ln 1 = 0$
   - Supposons la propriété vraie pour tout k tel que $1 \leqslant k \leqslant n-1$ et montrons là pour n :

$$L(n) \leqslant 1 + \frac{8}{n^2}\sum_{k=1}^{n-1} k\ln k \leqslant 1 + \frac{8}{n^2}\sum_{k=1}^{n-1}\int_k^{k+1} t\ln t\,dt \leqslant 1 + \frac{8}{n^2}\int_1^n t\ln t\,dt$$
$$\leqslant 1 + \frac{8}{n^2}\left(\frac{n^2}{2}\ln n - \frac{n^2}{4} + \frac{1}{4}\right) \leqslant 4\ln n - 1 + \frac{2}{n^2} \leqslant 4\ln n \quad \text{(puisque } n \geqslant 2)$$

2. Mieux ? De la formule $L(n) = \dfrac{n-1}{n} + \dfrac{2}{n^2}\sum_{k=0}^{n-1} kL(k)$, on déduit $L(n+1) = \dfrac{2n + n(n+2)L(n)}{(n+1)^2}$.
   Une petite simulation semble indiquer que la majoration de $L(n)$ par $4\ln n$ est généreuse. Cette même simulation semble plutôt indiquer que $\ln n < L(n) < 2\ln n$

3. Vraiment mieux ! Maple sait résoudre (incroyable mais vrai !) la récurrence établie ci-dessus :

```
> L := unapply( rsolve( {X(n) = (2*(n-1)+(n-1)*(n+1)*X(n-1))/n^2, X(1)=0},X),n);
> asympt( L(n),n);
```

$$L(n) = 2\,\frac{-1+(n+1)\left(\Psi(n+2)+\gamma\right)-2n}{n} = 2\ln n - 4 + 2\gamma + \frac{1+2\gamma+2\ln(n)}{n} + O\left(\frac{1}{n}\right)$$
$$= 2\ln n - 4 + 2\gamma + O\left(\frac{\ln n}{n}\right) \underset{n\to+\infty}{\sim} 2\ln(n) = O(2\ln n) \quad \text{avec, pour information,}$$

$$\Psi(x) = \frac{d\,\ln(\Gamma)}{dx}(x) = \frac{\Gamma'(x)}{\Gamma(x)} \quad ; \quad \Gamma(z) = \int_{t=0}^{+\infty} e^{-t}\,t^{z-1}\,dt \quad \text{et} \quad \gamma = \lim_{+\infty}\sum_{k=1}^{n}\frac{1}{k} - \ln n \approx 0.5772156649.$$

4. **En fait** , $\dfrac{n+1}{n+2}L(n+1) = \dfrac{n}{n+1}L(n) + \dfrac{2n}{(n+1)(n+2)}$ et on pose $U_n = \dfrac{n}{n+1}L(n)$ $\left(\underset{n\to+\infty}{\sim} L(n)\right)$.
   Alors, $U_{n+1} = U_n + \dfrac{2n}{(n+1)(n+2)}$, d'où $U_n = \sum_{k=0}^{n-1}\dfrac{2k}{(k+1)(k+2)} \underset{n\to+\infty}{\sim} 2\ln(n)$.

6 Nombre d'arbres binaires possédant n nœuds

Cette information peut être utile dans des calculs de complexité en moyenne.

On ne s'intéresse ici qu'à la structure des arbres binaires, indépendamment des informations contenues dans l'arbre (étiquettes de sommets), structure représentée par le type d'arbre binaire squelettique :

```
type arbre =
   | V
   | N of arbre * arbre
;;
```

Soit B_n le nombre d'arbre binaires possédants n nœuds. On a

$$\begin{cases} B_0 = 1; \quad B_1 = 1; \quad B_2 = 2; \quad B_3 = 5 \\ B_n = \displaystyle\sum_{k=0}^{n-1} B_k B_{n-1-k} \end{cases} \qquad (k \text{ nœuds à gauche et } n-1-k \text{ nœuds à droite de la racine})$$

Soit la série entière $\displaystyle\sum_{n\geqslant 0} B_n x^n$ et le produit de Cauchy de cette série par elle même, de terme général :

$$w_n x^n = \sum_{p+q=n} B_p B_q x^n = \sum_{p=0}^{n} B_p B_{n-p} x^n = B_{n+1} x^n$$

1. Supposons que la série $\displaystyle\sum_{n\geqslant 0} B_n x^n$ ait un rayon de convergence $R > 0$.

   Si S est la fonction somme de cette série, on aura pour $|x| < R$, $xS^2(x) = S(x) - 1$, ce qui donne $S(x) = \dfrac{1-\sqrt{1-4x}}{2x}$ (l'autre solution n'est pas prolongeable par continuité en 0).

   D'après les développements en série entière usuels, $\dfrac{1-\sqrt{1-4x}}{2x} = \displaystyle\sum_{n\geqslant 0} \frac{1}{n+1}\binom{2n}{n} x^n$ avec $R = \dfrac{1}{4}$

   On en déduit, par identification, que pour $n \geqslant 0$, $B_n = \dfrac{1}{n+1}\binom{2n}{n}$.

2. Il ne reste plus qu'à montrer que la série $\displaystyle\sum_{n\geqslant 0} B_n x^n$ a un rayon de convergence $R > 0$.

   Soit L la fonction de lecture d'un squelette d'arbre binaire, définie récursivement par

   $$L(V) = [f] \quad \text{et} \quad L\big(N(g,d)\big) = p :: L(g) @ L(d)$$

   L est une application de l'ensemble des squelettes d'arbres binaires vers l'ensemble des listes (suites) de symboles f et p et, l'application de L à

   - un arbre à n nœuds, donne une liste de $2n+1$ termes (n nœuds et $n+1$ feuilles) ;
   - deux arbres différents donnent deux listes différentes.

   Ainsi L est une application injective de l'ensemble Q_n des squelettes d'arbre binaire à n nœuds vers l'ensemble S_n des listes à $2n+1$ termes pris parmi f ou p.
   On a donc $B_n = card(Q_n) \leqslant card(S_n) = 2^{2n+1}$ et la série entière $\displaystyle\sum_{n\geqslant 0} 2^{2n+1} x^n$ ayant un rayon de convergence égal à $\dfrac{1}{4}$, on en déduit que la série $\displaystyle\sum_{n\geqslant 0} B_n x^n$ a un rayon de convergence $R \geqslant \dfrac{1}{4}$

Théorème 6.1.

Il y a $\dfrac{1}{n+1}\binom{2n}{n}$ *arbres binaires possédant n nœuds internes (et* $n+1$ *feuilles).*

Exercice : Ecrire en Caml la fonction L, sans utiliser la concaténation de liste @ .

7 Quelques familles d'arbres

7.1 Arbres binaires de recherche (ABR)

Arbre binaire homogène, tel que l'étiquette de tout nœud est

- supérieure ou égale à toute étiquette rencontrée dans le fils gauche
- inférieure ou égale (parfois strictement inférieure) à toute étiquette rencontrée dans le fils droit

On peut utiliser des arbres binaires de recherche pour gérer des dictionnaires etc ...

7.2 Arbres binaires de recherche équilibrés (ou quasi-équilibrés)

Dans de tels arbres, l'insertion et la suppression de nœuds) conservent à l'arbre une hauteur minimale ou quasi-minimale. Pour un arbre à n nœuds, cette hauteur est généralement un $O\big(\log_2(n)\big)$, ce qui fait que le coût de la recherche d'un élément est un $O\big(\log_2(n)\big)$.

- Arbres AVL. (Adel'son, Vel'skii, Landis, 1962)
- Arbres Rouge et Noir. (Bayer, Sleator, Tarjan), (et aussi Guibas, Sedgewick).
  - Chaque nœud est soit Rouge soit Noir et une feuille est toujours Noire.
  - Si un nœud est Rouge alors ses fils sont noirs.
  - Pour chaque nœud, tous les chemins conduisant à une feuille contiennent le même nombre de nœuds Noirs.
- Etc ...

7.3 B-Arbres. (Bayer, McCreight, 1972)

De tels arbres ont été conçus pour réaliser des accès rapides aux unités de disques.

- Un nœud x possède une étiquette composée de $n(x)$ valeurs (clés) et $n(x)+1$ fils,
- les clés d'un nœud, rangées par ordre croissant, déterminent les intervalles de valeurs des clés stockées dans les arbres fils,
- toutes les feuilles ont la même profondeur (égale à la hauteur de l'arbre),
- plus quelques autres propriétés ...

Cas particuliers : Arbres 2-3 (tout nœud possède 2 ou 3 fils), Arbres 2-3-4 (tout nœud possède 2, 3 ou 4 fils).

7.4 Tas et files de priorité

Un tas est un arbre binaire homogène (à feuilles vides), tel que l'étiquette de chaque nœud (sauf racine) est inférieure à celle de son père. Applications :

- **Tri en tas** d'un tableau de n éléments (tri en $O(n \ln n)$) ;
- **Files de priorités** : structure de données permettant de gérer un ensemble d'éléments en affectant chaque élément d'une valeur (clé), prise dans un ensemble ordonné, valeur qui représente la priorité accordée à l'élément de la file de priorité (donc son ordre de sortie).

7.5 Arbres de décisions

A chaque nœud est associé un test de décision, dont l'évaluation, dans un contexte donné, permet de choisir un fils de ce nœud.

7.6 Arbres d'expressions algébriques

La représentation d'expressions algébriques à l'aide d'arbres permet la réalisation du "calcul formel".

7.7 Autres ...

- Arbres 2-3 (Hopcroft 1970), généralisés par les B-Arbres. L'équilibre est maintenu en manipulant le degré des nœuds.
- Arbres déployés (Sleator, Tarjan, 1983).
- ...

$< \mathscr{F}\mathscr{I}\mathscr{N} >$