

# Structures de données arborescentes

## 1 Motivations

### 1.1 Un arbre pour un objet

Pour des éléments d'un ensemble construit par induction, il est approprié d'utiliser une représentation par arbre puisque ces objets ont intrinsèquement une structure arborescente

**Exemple :** Les expressions booléennes, arithmétiques, de type.

### 1.2 Un arbre pour une collection d'objet

On cherche à stocker une collection d'objets sans multiplicité, et dont l'ordre relatif n'est pas significatif (comme dans le cas d'un ensemble).

On suppose que tous les éléments sont du type `elem`, et qu'ils sont identifiés de manière unique par une clé, c'est à dire une sous-partie permettant l'identification.

On a alors les méthodes suivantes :

- `creer_ens_vide` :  $() \rightarrow \text{ens}$
- `ajoute_elem` :  $\text{ens} \times \text{elem} \rightarrow \text{ens}$
- `est_ens_vide` :  $\text{ens} \rightarrow \text{bool}$
- `supprime_elem` :  $\text{ens} \times \text{clé} \rightarrow \text{ens}$
- `appartient` :  $\text{ens} \times \text{elem} \rightarrow \text{bool}$
- `trouve_elem` :  $\text{ens} \times \text{clé} \rightarrow \text{elem}$

**Remarque :** On peut aussi imaginer des fonctions `ajoute_elem` et `supprime_elem` qui modifieraient directement l'ensemble donné en entrée, plutôt que de renvoyer un nouvel ensemble.

### 1.3 Implémentations

On peut stocker les éléments dans une liste. On peut aussi associer chaque élément à une clé (pour un ensemble de caractères par exemple, leurs valeurs ascii), qui permet de les comparer rapidement. Si l'on peut ordonner ces clés, on peut alors classer les éléments par ordre croissant dans un tableau.

Opération	Complexité (Liste)	Complexité (Tableau ordonné)
<code>appartient</code>	$\theta(n)$	$\theta(\log(n))$ (dichotomie)
<code>ajoute_elem</code>	$\theta(1)$	$\theta(n)$
<code>supprime_elem</code>	$\theta(n)$	$\theta(n)$
<code>trouve_elem</code>	$\theta(n)$	$\theta(\log(n))$

**Remarque :** On voit selon le contexte qu'une certaine implémentation sera plus efficace qu'une autre : si on doit faire beaucoup d'insertions sans trop chercher d'éléments, le plus efficace sera la liste. Par contre, si on n'insère que rarement des éléments mais que l'on est souvent amené à chercher dans les éléments, le tableau trié sera à préférer.

Il existe cela dit une structure qui permet d'avoir une complexité d'ajout / suppression et de recherche en  $\theta(\log(n))$ , sous certaines conditions : il s'agit des **arbres binaires de recherche (ABR)**.

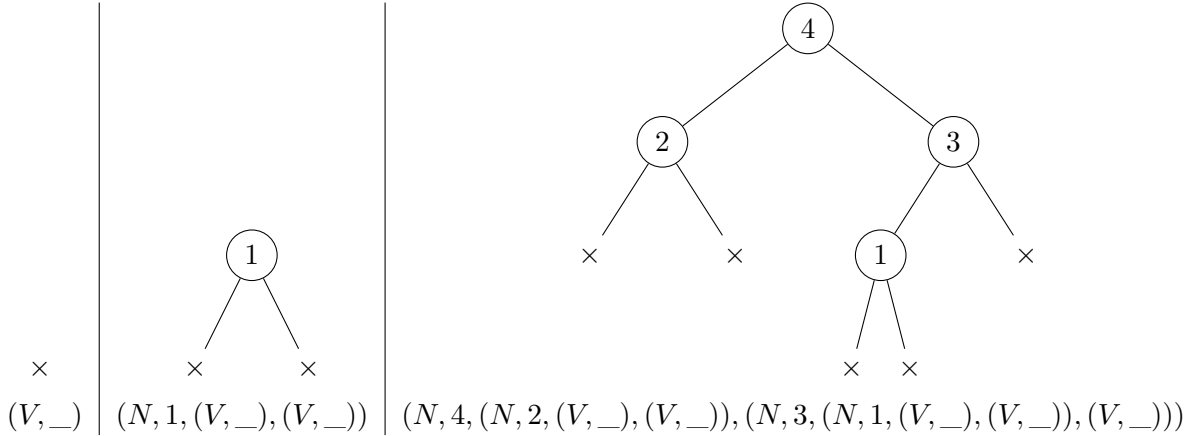
## 2 Arbre binaires

### 2.1 Ensemble $\mathcal{A}_B$

Soit  $\mathcal{S}$  un ensemble. On définit l'ensemble des arbres binaires étiquetés par  $\mathcal{S}$  comme l'ensemble construit par induction à partir de ces deux règles :

- $V \Big|_{\{-\}}^0$
- $N \Big|_{\mathcal{S}}^2$

**Exemple :** Soit  $\mathcal{S} = \{1, 2, 3, 4\}$ .



### 2.2 Feuilles

On fixe désormais  $\mathcal{S}$  un ensemble d'étiquettes. On notera par ailleurs  $N(x, g, d) = (N, x, g, d)$  et  $V = (V, \_)$  pour alléger l'écriture.

On dit qu'un arbre  $t \in \mathcal{A}_B(\mathcal{S})$  est réduit à une feuille ssi il existe  $x \in \mathcal{S}$  tel que  $t = N(x, V, V)$ .

**Propriété :** Pour la relation d'ordre  $\leq$  associée à la définition inductive de  $\mathcal{A}_B(\mathcal{S})$ , on a :

1.  $V$  est le seul élément minimal de  $(\mathcal{A}_B(\mathcal{S}), \leq)$ .
  2.  $t$  élément minimal de  $\mathcal{A}_B(\mathcal{S}) \setminus \{V\} \Leftrightarrow t$  est réduit à une feuille.
- ▷ Le 1) découle du fait que  $V$  soit le seul cas de base des règles d'induction de  $\mathcal{A}_B(\mathcal{S})$ .
- ▷ Soit  $t$  minimal dans  $\mathcal{A}_B(\mathcal{S}) \setminus \{V\}$ . Il existe par définition  $x \in \mathcal{S}$  et  $(g, d) \in \mathcal{A}_B(\mathcal{S})^2$  tels que  $t = N(x, g, d)$ . Comme  $g \leq t$  et  $d \leq t$ , et d'autre part  $g \neq t$  et  $d \neq t$ , on en déduit que  $g < t$  et  $d < t$ . Par minimalité de  $t$  dans  $\mathcal{A}_B(\mathcal{S}) \setminus \{V\}$ , on en déduit que  $g = d = V$ , donc que  $t$  est réduit à une feuille.

### 2.3 Chemins

Considérons l'alphabet  $\Sigma = \{0, 1\}$ . On définit par induction sur  $\mathcal{A}_B(\mathcal{S})$  l'ensemble des chemins admissibles d'un arbre binaire  $t$ , noté  $\text{ch}(t)$ , par :

$$\forall t \in \mathcal{A}_B(\mathcal{S}) : \quad \text{ch}(t) = \begin{cases} \emptyset & \text{si } t = V \\ \{\varepsilon\} \cup \{0.\text{ch}(g)\} \cup \{1.\text{ch}(d)\} & \text{si } t = N(x, g, d) \end{cases}$$

Soit  $t \in \mathcal{A}_B(\mathcal{S})$ . Un **noeud**  $\mathcal{N}$  de  $t$  est un élément de  $\text{ch}(t)$ . Sa profondeur, notée  $|\mathcal{N}|$  est alors sa longueur en tant que mot de  $\Sigma^*$ . La taille de  $t$ , notée  $s(t)$ , est alors le nombre de noeuds de  $t$ , autrement dit  $\text{cardch}(t)$ .

**Remarque :** Un chemin admissible décrit la "position" d'un "noeud" dans l'arbre.

**Exercice 1:** Donner une définition inductive de  $s(t)$ .

On appelle **étiquetage** d'un arbre non vide la fonction qui à un noeud de l'arbre associe son étiquette.

Formellement, l'étiquetage est défini inductivement comme suit (on note  $\mathcal{E}$  l'ensemble des étiquetages des arbres, c'est à dire des fonctions d'une partie de  $\Sigma^*$  dans  $\mathcal{S}$ ) :

$$\text{etiq} \left( \begin{array}{l} \mathcal{A}_B(\mathcal{S}) \setminus \{V\} \rightarrow \mathcal{E} \\ N(x, V, V) \mapsto \left( \begin{array}{l} \text{ch}(x, V, V) = \{\varepsilon\} \rightarrow \mathcal{S} \\ \varepsilon \mapsto x \end{array} \right) \\ N(x, g, V) \mapsto \left( \begin{array}{l} \text{ch}(N, x, g, V) \rightarrow \mathcal{S} \\ \varepsilon \mapsto x \\ 0 \cdot u \mapsto \text{etiq}(g)(u) \end{array} \right) \\ N(x, V, d) \mapsto \left( \begin{array}{l} \text{ch}(N, x, V, d) \rightarrow \mathcal{S} \\ \varepsilon \mapsto x \\ 1 \cdot u \mapsto \text{etiq}(d)(u) \end{array} \right) \\ N(x, g, d) \mapsto \left( \begin{array}{l} \text{ch}(N, x, g, d) \rightarrow \mathcal{S} \\ \varepsilon \mapsto x \\ 0 \cdot u \mapsto \text{etiq}(g)(u) \\ 1 \cdot u \mapsto \text{etiq}(d)(u) \end{array} \right) \end{array} \right)$$

Soit  $t \in \mathcal{A}_B(\mathcal{S})$ ,  $\text{etiq}(t)$  est l'étiquetage de  $t$ . Si  $n \in \text{ch}(t)$ , alors l'étiquette de  $n$  dans  $t$  est  $(\text{etiq}(t))(n)$

## 2.4 Vocabulaire

Soit  $t \in \mathcal{A}_B(\mathcal{S})$ . Soit  $n \in \text{ch}(t)$ .

- $n$  est une **feuille** de  $t$  si et seulement si pour tout  $u \in \Sigma^*$ ,  $n \cdot u \in \text{ch}(t) \Rightarrow u = \varepsilon$
- $n$  est **racine** de  $t$  si et seulement si  $n = \varepsilon$
- $n$  est un **noeud interne** de  $t$  si et seulement si  $n$  n'est pas une feuille.

Soient  $n$  et  $m$  deux chemins admissibles pour  $t$  un arbre binaire non vide.  $m$  est le **fil gauche** (resp. **fil droit**) de  $n$  si et seulement si  $m = n \cdot 0$  (resp.  $m = n \cdot 1$ ). Dans les deux cas,  $n$  est alors le **père** de  $m$ .

On dit que  $m$  est un **descendant** de  $n$  si et seulement si il existe  $u \in \Sigma^*$  tel que  $m = n \cdot u$ . Dans ce cas  $n$  est un **ascendant** de  $m$ .

**Remarque :** Les feuilles sont les noeuds sans enfant, et la racine est le seul noeud sans père.

Soit  $(n_i)_{i \in \llbracket 0, k \rrbracket} \in \text{ch}(t)^{k+1}$ . On dit que  $(n_i)$  est une **branche** de  $t$  si et seulement si  $n_0 = \varepsilon$  et  $n_i$  est le père de  $n_{i+1}$  pour tout  $i \in \llbracket 0, k \rrbracket$ .

Soient  $t$  et  $t'$  deux arbres binaires sur  $\mathcal{S}$ . On dit que  $t'$  est le **sous-arbre droit** (resp **sous-arbre gauche**) de  $t$  si et seulement si il existe  $g \in \mathcal{A}_B(\mathcal{S})$  (resp.  $d \in \mathcal{A}_B(\mathcal{S})$ ) tel que  $t = N(x, g, t')$  (resp.  $t = N(x, t', d)$ ). On dit que  $t'$  est un **sous-arbre** de  $t$  si et seulement si  $t' \leq t$ .

**Remarque :** Notons qu'un arbre binaire peut-être sous-arbre d'un autre tout en étant ni un sous-arbre droit, ni un sous-arbre gauche.

## 2.5 Hauteur

On définit par induction la hauteur  $h(t)$  d'un arbre binaire  $t \in \mathcal{A}_B(\mathcal{S})$  comme valant -1 si  $t = V$  et  $1 + \max(h(g), h(d))$  si  $t = N(x, g, d)$ .

**Propriété :** Soit  $t \in \mathcal{A}_B(\mathcal{S}) \setminus \{V\}$ . Alors  $h(t) = \max_{n \in \text{ch}(t)} \text{prof}(n)$ .  $h(t) + 1$  est alors la longueur maximale d'une branche.

- ▷ Montrons-le par induction sur  $t$  : c'est bien le cas pour  $t = N(x, V, V)$  car  $h(t) = 0$ , le seul chemin admissible pour cet arbre est  $\varepsilon$  qui est de longueur 0, et la seule branche de  $t$  est alors  $(\varepsilon)$ , qui est de taille 1.
- ▷ Soit  $t = N(x, g, d) \in \mathcal{A}_B(\mathcal{S})$  avec  $g$  et  $d$  deux arbres binaires non vides vérifiant la propriété. On a

$$\max_{n \in \text{ch}(t)} \text{prof}(n) = \max \left( \max_{0 \cdot n \in \text{ch}(t)} \text{prof}(n), \max_{1 \cdot n \in \text{ch}(t)} \text{prof}(n), 0 \right)$$

en séparant les chemins admissibles de  $t$  selon leur première lettre. Par définition des chemins admissibles et de la profondeur, on a alors

$$\max_{n \in \text{ch}(t)} \text{prof}(n) = \max(1 + \max_{n \in \text{ch}(g)} \text{prof}(n), 1 + \max_{n \in \text{ch}(d)} \text{prof}(n), 1)$$

soit, par hypothèse sur  $g$  et  $d$ ,

$$\max_{n \in \text{ch}(t)} \text{prof}(n) = 1 + \max(h(g), h(d), 0) = 1 + \max(h(g), h(d)) = h(t)$$

car  $g$  et  $d$  ne sont pas vides et ont donc une hauteur plus grande que 0. On établit le résultat sur la longueur maximale des branches de la même manière.

- ▷ Les cas  $t = N(x, g, V)$  et  $t = N(x, V, d)$  se traitent de la même manière (la séparation des maximums donne alors un maximum d'un ensemble vide, soit  $-\infty$ , qui est neutre pour le maximum, ce qui traduit l'inexistence de branches / noeuds à droite ou à gauche de la racine).

**Propriété :** Pour  $t \in \mathcal{A}_B(\mathcal{S})$ , on a  $h(t) + 1 \leq s(t) \leq 2^{h(t)+1} - 1$ .

- ▷ Montrons-le par induction sur  $t$  : pour  $t = V$ , on a  $h(t) + 1 = 0$ ,  $s(t) = 0$  et  $2^{h(t)+1} - 1 = 0$ .  
 ▷ Soit  $t = N(x, g, d) \in \mathcal{A}_B(\mathcal{S})$ , où  $g$  et  $d$  respectent la propriété énoncée. Alors

$$h(t) + 1 = 2 + \max(h(d), h(g)) \leq 2 + h(d) + h(g)$$

$$1 + (h(g) + 1) + (h(d) + 1) = 3 + h(g) + h(d) \leq 1 + s(g) + s(d)$$

$$s(g) + s(d) + 1 \leq 2^{h(t)+1} - 1 + 2^{h(d)+1} \leq 2^{\max(h(t), h(d))+1} - 1 = 2^{h(t)+1} - 1$$

En combinant ces inégalités, il vient le résultat attendu.

## 2.6 Parcours

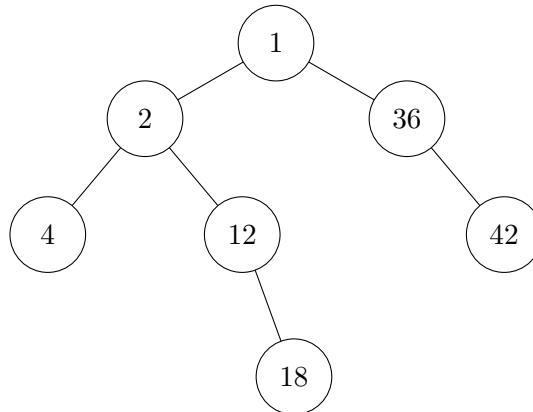
Soit  $t \in \mathcal{A}_B(\mathcal{S}) \setminus \{V\}$ .  $(a_i)_{i \in \llbracket 1, s(t) \rrbracket}$  est un **parcours** de  $t$  si et seulement si il existe  $\varphi \in \mathcal{F}(\text{ch}(t), \llbracket 1, s(t) \rrbracket)$  bijective telle que  $\forall n \in \text{ch}(t), a_{\varphi(n)} = \text{eti}(t)(n)$ .

Pour  $n \in \text{ch}(t)$  on note

- $\mathcal{G}(n) = \{n \cdot 0 \cdot u \mid u \in \Sigma^*\} \cap \text{ch}(t)$  l'ensemble des descendants gauches de  $n$  ;
- $\mathcal{D}(n) = \{n \cdot 1 \cdot u \mid u \in \Sigma^*\} \cap \text{ch}(t)$  l'ensemble des descendants droits de  $n$ .

Un parcours  $(\text{eti}(t)(\varphi^{-1}(i)))_{i \in \llbracket 1, s(t) \rrbracket}$  (où  $\varphi$  est une bijection de  $\text{ch}(t)$  dans  $\llbracket 1, s(t) \rrbracket$ ) est dit **préfixe** (resp. **postfixe**, **infixe**) si et seulement si pour tout  $n \in \text{ch}(t)$ , pour tout  $g \in \mathcal{G}(n)$  et  $d \in \mathcal{D}(n)$ , on a  $\varphi(n) \leq \varphi(g) \leq \varphi(d)$  (resp.  $\varphi(g) \leq \varphi(d) \leq \varphi(n)$ ,  $\varphi(g) \leq \varphi(n) \leq \varphi(d)$ ).

**Exemple :** Pour l'arbre suivant (les noeuds vides n'ont pas été représentés) :



- Un parcours préfixe est 1 ; 2 ; 4 ; 12 ; 18 ; 36 ; 42 ;
- Un parcours postfixe est 4 ; 18 ; 12 ; 2 ; 42 ; 36 ; 1 ;
- Un parcours infixe est 4 ; 2 ; 12 ; 18 ; 1 ; 36 ; 42.

**Propriété :** Il y a unicité des parcours préfixes, infixes et postfixes.

▷ En exercice.

On définit aussi le **parcours en largeur**, pour lequel  $(\text{prof}(\varphi^{-1}(i)))_{i \in \llbracket 1, s(t) \rrbracket}$  doit être croissante, et de parcours en profondeur (hors programme dans le cadre des arbres, mais cela suit le même principe que les parcours en profondeur dans les graphes).

### 3 Arbres binaires de recherches (ABR)

Dans cette partie,  $E$  désigne un ensemble muni d'une relation d'ordre totale, notée  $\preceq$ .

Soit  $t \in \mathcal{A}_B(E) \setminus \{V\}$ . On pose  $e = \text{etiq}(t)$ .  $t$  est un arbre binaire de recherche (abgrégé ABR) si et seulement si pour tout  $n \in \text{ch}(t)$ ,  $\max_{g \in \mathcal{G}(n)} e(g) \leq e(n) < \min_{d \in \mathcal{D}(n)} e(n)$ . Par convention, l'arbre vide est un arbre de recherche.

#### 3.1 Recherche d'éléments

Profitons de la structure ordonnée de l'ABR. On procède par dichotomie pour la recherche d'un élément (d'une étiquette).

---

**Algorithme 1 :** Recherche d'élément dans un ABR

---

**Entrées :**  $t \in \mathcal{A}_B(E)$  un ABR à étiquettes dans  $E$ ,  $e \in E$  l'élément à rechercher dans  $t$ .

**Sorties :** Vrai s'il existe un noeud dans  $t$  ayant pour étiquette  $e$ , faux sinon.

```

1 si  $t = V$  alors
2   | Renvoyer faux
3 sinon
4   | Poser  $t = N(x, g, d)$  ;
5   | si  $e = x$  alors
6   |   | Renvoyer vrai
7   | sinon
8   |   | si  $e < x$  alors
9   |   |   | Rechercher  $e$  dans  $g$ 
10  |   |   | sinon
11  |   |   | Rechercher  $e$  dans  $d$ 
12  |   | fin
13  | fin
14 fin
```

---

( $\mathbb{B}$  désigne l'ensemble des booléens : vrai ou faux)

**Propriété :** Pour tout noeud  $n$  d'un arbre binaire de recherche  $t \in \mathcal{A}_B(E)$ , s'il n'existe pas d'autre noeud dans  $t$  ayant la même étiquette que  $n$ , alors le nombre de comparaisons (entre éléments de  $E$ ) effectuées lors de la recherche de  $\text{etiq}(t)(n)$  dans  $t$  vaut  $2\text{prof}(n) + 1$ .

▷ Montrons-le par récurrence sur la profondeur du noeud  $n$  : le prédicat à prouver est, pour  $k \in \mathbb{N}$

$\mathcal{P}(k)$  : pour tout noeud de taille  $k$  d'un arbre  $t \in \mathcal{A}_B(E)$ , s'il n'existe pas d'autre noeud dans  $t$  même étiquette que  $n$ , alors le nombre de comparaison lors de la recherche de  $e = \text{etiq}(t)(n)$  dans  $t$  vaut  $2k + 1$ .

▷ Pour  $k = 0$ , on a nécessairement  $n = \varepsilon$ , ce qui implique que l'arbre sur lequel on appelle la fonction n'est pas vide, et que sa racine est étiquetée par  $e$  : ainsi, on effectue une seule comparaison (1.5) avant de renvoyer vrai. D'où  $\mathcal{P}(0)$ .

▷ Soit  $k \in \mathbb{N}$ . Supposons  $\mathcal{P}(k)$ . Soit  $n$  un chemin de profondeur  $k + 1$  d'un arbre binaire de recherche  $t$ . Lors de l'appel initial à la recherche, le test d'égalité ligne 5 échoue (sinon on aurait deux noeuds étiquetés par  $e$ , ce qui est exclu par hypothèse). On engendre donc un appel au sous-arbre gauche ou

droit (cela dépend du premier caractère de  $n = 0 \cdot m$  ou  $n = 1 \cdot m$ ) après une seconde comparaison. Dans ce sous-arbre, le noeud étiqueté par  $e$  dans l'arbre initial est  $m$  (par définition inductive de l'étiquetage), et est toujours le seul à être étiqueté par  $e$ . Comme  $m$  est de profondeur  $k$ , on sait d'après  $\mathcal{P}(k)$  que la suite de la recherche va alors engendrer  $2k + 1$  nouvelles comparaisons. Sommées avec les comparaisons déjà effectuées, on obtient un total de  $2k + 1 + 2 = 2(k + 1) + 1$  comparaisons, d'où  $\mathcal{P}(k + 1)$ .

**Corrolaire :** *Pour tout  $e \in E$ , la recherche de  $x$  dans  $t$  engendrera au plus  $2h(t) + 2$  comparaisons.*

- ▷ Si  $e$  n'est pas dans  $t$ , on le prouve par une induction élémentaire sur  $t$ . Sinon, soit  $n$  un noeud de profondeur minimale parmi tous les noeuds étiquetés par  $e$ . En reprenant la preuve précédente en utilisant la minimalité de la profondeur de  $n$  plutôt que le fait que  $n$  soit le seul noeud à être étiqueté par  $x$ , on prouve que l'algorithme effectue  $2\text{prof}(n) + 1$  comparaisons. D'après la remarque sur le lien entre profondeur des noeuds est hauteur, le nombre total de comparaisons effectuées est bien majoré par  $2h(t) + 1$ .

## 3.2 Ajout en feuille

---

**Algorithme 2 :** Ajout en feuille dans un arbre binaire de recherche

---

**Entrées :**  $t \in \mathcal{A}_B(E)$  un ABR,  $e \in E$  un élément à ajouter dans  $t$

**Sorties :**  $t'$  un ABR contenant tous les éléments de  $t$  (avec multiplicité) et  $e$ .

```

1 si  $t = V$  alors
2   | Renvoyer  $N(e, V, V)$ 
3 sinon
4   | Poser  $t = N(x, g, d)$  ;
5   | si  $e \leq x$  alors
6     | Ajouter  $e$  en feuille à  $g$ . On note  $g'$  l'arbre obtenu ;
7     | Renvoyer  $N(x, g', d)$ 
8   | sinon
9     | Ajouter  $e$  en feuille à  $d$ . On note  $d'$  l'arbre obtenu ;
10    | Renvoyer  $N(x, g, d')$ 
11  fin
12 fin
```

---

**Propriété :** *Si  $t$  est un ABR, l'arbre obtenu en ajoutant à  $t$  un élément en feuille est toujours un ABR*

- ▷ En exercice (par induction sur  $t$ ).

**Propriété :** *La complexité temporelle de l'ajout en feuille est en  $O(h(t))$ .*

- ▷ On montre par induction sur  $t$  que l'ajout en feuille engendre au plus  $h(t) + 1$  comparaisons entre éléments de  $E$

### **3.3 Suppression**

### **3.4 Limitation de la hauteur**

## **4 Arbres 2-3-4**

### **4.1 Définitions**

### **4.2 Scission d'un 4-noeud**

### **4.3 Insertion dans un arbre 2-3-4**

### **4.4 Suppression dans un arbre 2-3-4**

## **5 Arbres rouge-noir**

## **6 Hachage**

## **7 Implémentation des tas**

### **7.1 Introduction**

### **7.2 Opérations en pseudo-code**