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 : () -> ens`
- `est_ens_vide : ens -> bool`
- `appartient : ens×elem -> bool`
- `ajoute_elem : ens×elem -> ens`
- `supprime_elem : ens×clé -> ens`
- `trouve_elem : ens×clé -> 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)
`appartient`	$\theta(n)$	$\theta(\log(n))$ (dichotomie)
`ajoute_elem`	$\theta(1)$	$\theta(n)$
`supprime_elem`	$\theta(n)$	$\theta(n)$
`trouve_elem`	$\theta(n)$	$\theta(\log(n))$

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 ammené à chercher dans les éléments, le tableau trié sera à préférer.

ll 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)**.

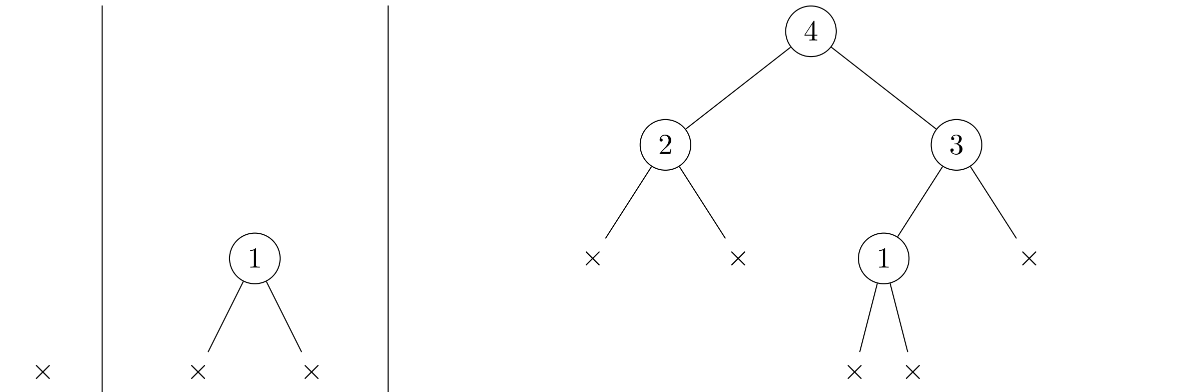
2 Arbre binaires

2.1 Ensemble $\mathscr{A}_B$

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 :

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

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



$(V, \_)$ | $(N, 1, (V, \_), (V, \_))$ | $(N, 4, (N, 2, (V, \_), (V, \_)), (N, 3, (N, 1, (V, \_), (V, \_)), (V, \_)))$

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 $t \in \mathscr{A}_B(\mathcal{S})$ 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 $\mathscr{A}_B(\mathcal{S})$, on a :*

1. *V est le seul élément minimal de $(\mathscr{A}_B(\mathcal{S}), \leq)$.*
2. *t élément minimal de $\mathscr{A}_B(\mathcal{S}) \setminus \{V\} \Leftrightarrow t$ est réduit à une feuille.*

- ▷ Le 1) découle du fait que V soit le seul cas de base des règles d'induction de $\mathscr{A}_B(\mathcal{S})$.
- ▷ Soit t minimal dans $\mathscr{A}_B(\mathcal{S}) \setminus \{V\}$. Il existe par définition $x \in \mathcal{S}$ et $(g, d) \in \mathscr{A}_B(\mathcal{S})^2$ 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 $\mathscr{A}_B(\mathcal{S}) \setminus \{V\}$, 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 $\mathscr{A}_B(\mathcal{S})$ l'ensemble des chemins admissibles d'un arbre binaire t, noté $\mathrm{ch}(t)$, par :

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

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

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 Σ^* dans $\mathcal{S}$) :

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

Soit $t \in \mathscr{A}_B(\mathcal{S})$, 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 $t \in \mathscr{A}_B(\mathcal{S})$. 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 **fils gauche** (resp. **fils 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 $(n_i)_{i\in[\![0,k]\!]} \in \text{ch}(t)^{k+1}$. 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 $i \in [\![0,k[\![$.

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 $g \in \mathscr{A}_B(\mathcal{S})$ (resp. $d \in \mathscr{A}_B(\mathcal{S})$) 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 $t \in \mathscr{A}_B(\mathcal{S})$ comme valant -1 si $t = V$ et $1 + \max(h(g), h(d))$ si $t = N(x,g,d)$.

Propriété : *Soit $t \in \mathscr{A}_B(\mathcal{S})\setminus\{V\}$. Alors $h(t) = \max_{n\in ch(t)} \text{prof}(n)$. $h(t)+1$ 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 ε qui est de longueur 0, et la seule branche de t est alors (ε), qui est de taille 1.

▷ Soit $t = N(x,g,d) \in \mathscr{A}_B(\mathcal{S})$ avec g et d deux arbres binaires non vides vérifiant la propriété. On a

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

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

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

soit, par hypothèse sur g et d,

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

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'inexistance de branches / noeuds à droite ou à gauche de la racine).

Propriété : *Pour $t \in \mathscr{A}_B(\mathcal{S})$, 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 $t = N(x, g, d) \in \mathscr{A}_B(\mathcal{S})$, 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 $t \in \mathscr{A}_B(\mathcal{S}) \setminus \{V\}$. $(a_i)_{i\in[\![1,s(t)]\!]}$ est un **parcours** de t si et seulement si il existe $\varphi \in \mathcal{F}(\mathrm{ch}(t), [\![1, s(t)]\!])$ bijective telle que $\forall n \in \mathrm{ch}(t), a_{\varphi(n)} = \mathrm{etiq}(t)(n)$.

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

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

Un parcours $(\mathrm{etiq}(t)(\varphi^{-1}(i)))_{i\in[\![1,s(t)]\!]}$ (où φ est une bijection de $\mathrm{ch}(t)$ dans $[\![1, s(t)]\!]$) est dit **préfixe** (resp. postfixe, infixe) si et seulement si pour tout $n \in \mathrm{ch}(t)$, pour tout $g \in \mathscr{G}(n)$ et $d \in \mathscr{D}(n)$, on a $\varphi(n) \leq \varphi(g) \leq \varphi(d)$ (resp. $\varphi(g) \leq \varphi(d) \leq \varphi(n)$ et $\varphi(g) \leq \varphi(n) \leq \varphi(d)$).

Exemple : Pour l'arbre suivant :