Structure de données arborescentes

Guillaume Rousseau
MP2I Lycée Pierre de Fermat
guillaume.rousseau@ens-lyon.fr

27 mars 2025

1 Arbres

A Introduction

Les arbres permettent de représenter des situations et des objets hiérarchiques. Par exemple :

— Les résultats d'un tournoi

— Les dossiers d'un ordinateur

— Les expressions arithmétiques

— Les programmes OCaml

Les arbres permettent également d'implémenter de manière efficace certaines structures de données comme les dictionnaires.

Définition 1. Soit E un ensemble d'éléments appelés ***étiquettes***. On défini inductivement l'ensemble des arbres binaires étiquetés par E, noté $\mathcal{A}_E$ comme suit :

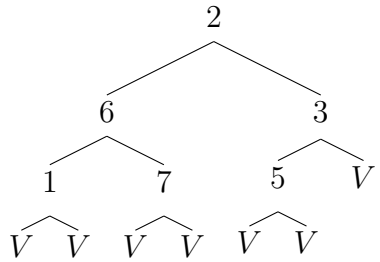
— L'arbre vide V est un arbre étiqueté par E : $V \in \mathcal{A}_E$.

— Pour $x \in E, r \in \mathbb{N}^*$, et $G, D \in \mathcal{A}_E$ deux arbres, $N(e, G, D) \in \mathcal{A}_E$.

Autrement dit, on s'intéresse à l'ensemble correspondant au type OCaml suivant :

```
1 type 'a arbre = V | N of 'a * 'a arbre * 'a arbre
```

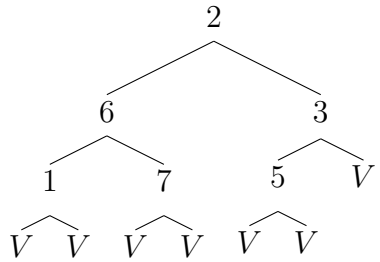
On représente généralement les arbres sous forme de schémas expliquant comment enchaîner les N et les V pour les construire. Par exemple, considérons l'arbre suivant :



Ce que l'on noterait $N(2, N(6, N(1, V, V), N(7, V, V)), N(3, N(5, V, V), V))$

B Noeuds

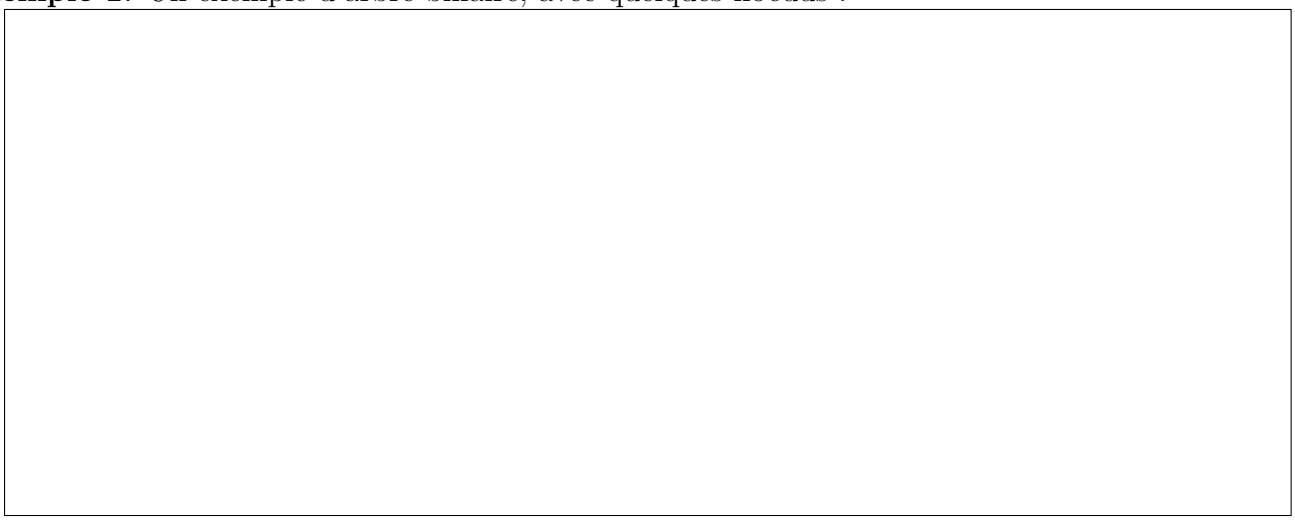
On peut repérer les différents éléments d'un arbre en décrivant le chemin permettant de les atteindre depuis la racine. Par exemple, considérons l'arbre suivant.



L'élément 7 se trouve à gauche puis à droite. En identifiant l'ensemble $\{\textbf{gauche}, \textbf{droite}\}$ à $\{0, 1\}$, on peut dire que la position 01 de l'arbre contient 7.

Définition 2. On appelle **noeud** d'un arbre binaire A toute suite finie de $\{0, 1\}$ correspondant à une position valide de A. L'étiquette d'un noeud est la valeur stockée à cette position dans l'arbre. La ***profondeur*** d'un noeud est sa longueur en tant que suite finie. La ***racine*** d'un arbre est la suite vide, de longueur 0 (souvent noté ε).

Exemple 1. Un exemple d'arbre binaire, avec quelques noeuds :



Définition 3. On appelle ***taille*** de A le nombre de noeuds de A.
On appelle ***hauteur*** de A la profondeur de son noeud le plus profond.
Par convention, on dit que **la hauteur de l'arbre vide est -1.**

Exemple 2. Quelques arbres, avec leur hauteur et leur taille :

Exercice 1. Compléter les deux fonctions suivantes permettant de calculer la taille et la hauteur d'un arbre en OCaml :

```
1 let rec taille (a: 'a arbre) : int =
2   match a with
3   | V -> 0
4   | N(x, g, d) ->
5
6 let rec hauteur (a: 'a arbre) : int =
7   match a with
8   | V ->
9   | N(x, g, d) ->
```

Définition 4. Soit A un arbre et $u = u_1 \dots u_k$ un noeud de A.

— Si $k > 0$ alors u n'est pas la racine et a pour **parent** le noeud $u_1 \dots u_k$.

— u admet deux **enfants** : $u_1 \dots u_k 0$ et $u_1 \dots u_k 1$ (respectivement appelés enfant gauche et enfant droit).

Le parent d'un noeud est donc le noeud situé directement au dessus de lui, et ses deux enfants sont les deux noeuds situés directement en dessous.

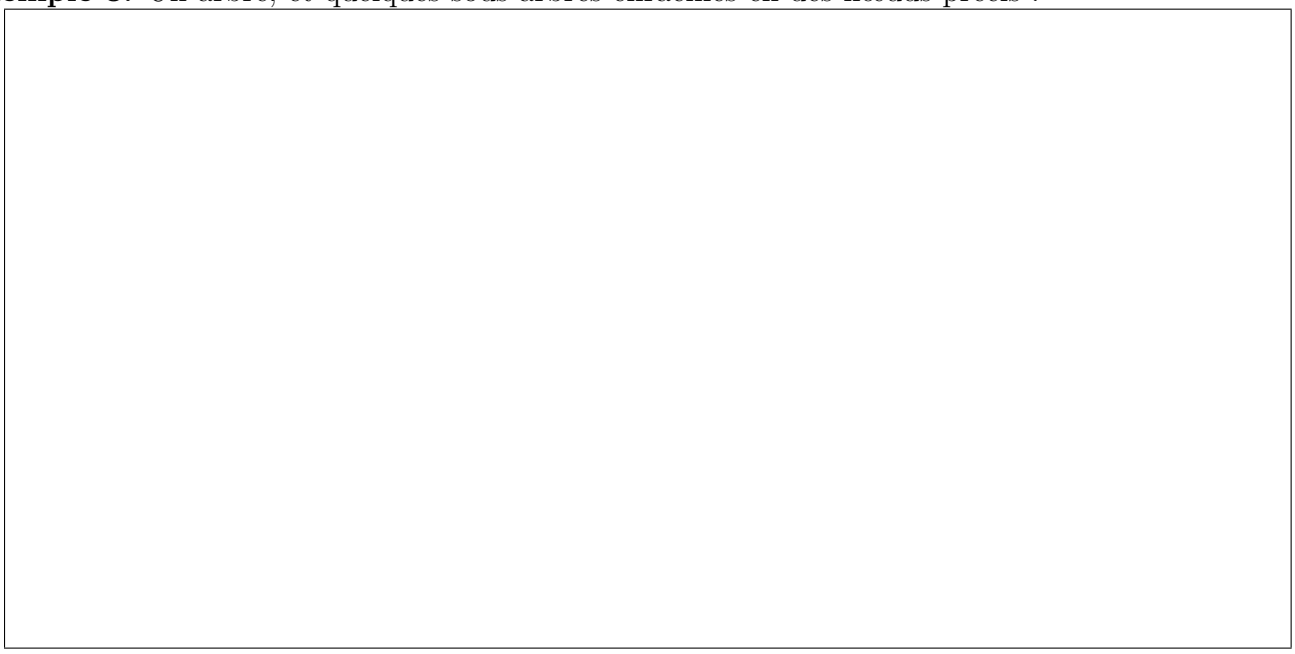
Définition 5. Soit A un arbre, et u un noeud de A. Le sous-arbre de A enraciné en u est l'arbre$A$ commençant à la position marquée par u. On peut le définir par récurrence sur la longueur de u :

— Le sous-arbre de A enraciné en ε est A lui-même

— Pour u un noeud et $N(x, G, D)$ un arbre non-vide, le sous-arbre de A enraciné en $0u$ est le sous-arbre de G enraciné en u.

— Pour u un noeud et $N(x, G, D)$ un arbre non-vide, le sous-arbre de A enraciné en $1u$ est le sous-arbre de D enraciné en u.

Exemple 3. Un arbre, et quelques sous-arbres enracinés en des nœuds précis :



En OCaml, on pourra écrire :

```
1 (* true <-> droite *)
2 let rec sous_arbre (a: 'a arbre) (u: bool list) =
3   match a, u with
4   | V, _ -> failwith "arbre vide"
5   | _, [] -> a
6   | N(x, g, d), b :: q ->
7     if b then sous_arbre d q
8     else      sous_arbre g q
```

C Arbres stricts

Définition 6. On dit qu'un arbre est binaire strict si tous ses nœuds ont exactement 0 ou 2.

Exemple 4. Deux arbres binaires : un strict et un non-strict :

Notons que dans un arbre binaire strict, les V n'apportent aucune information : on peut les omettre des schémas et déduire où ils sont placés.
En général, lorsque l'on considère des arbres stricts, on pose une définition légèrement différente des arbres, faisant disparaître le V.

Exercice 2.

On définit $\mathcal{AS}_E$ l'ensemble des arbres binaires stricts étiquetés par E par induction :

— Pour $e \in E$, $\mathbf{F}(e) \in \mathcal{AS}_E$: c'est une **feuille** étiquetée par e.

— Pour $e \in E$ et $g, d \in \mathcal{AS}_E$, $\mathbf{N}(e, g, d) \in \mathcal{AS}_E$: c'est un **nœud interne** étiqueté par e et ayant d comme enfants.

Notons que cet ensemble ne contient pas l'arbre vide.

Question 1. Dessiner l'arbre binaire strict suivant :

```
1 N(1, N(2, N(3, F 4, F 5), F 6), N(7, F 8, F 9))
```

Question 2. Combien l'arbre précédent a-t-il de noeuds internes ? de feuilles ? de noeuds ? Quelle est sa hauteur ?

Question 3. Définir une fonction inductive (i.e. récursive sur les arbres) permettant de calculer la taille d'un arbre strict. Faire de même pour compter le nombre de feuilles et de nœuds internes.

Question 4. Définir une fonction inductive permettant de calculer la hauteur d'un arbre strict, et donner sa complexité en fonction de n la taille de l'arbre.

Question 5. On considère un arbre binaire strict, a. On note h sa hauteur, i son nombre de nœuds internes, f son nombre de feuilles et n son nombre de nœuds. Montrer par induction structurelle sur les arbres stricts les relations suivantes entre h, f, i et n :

1. $n = f + i$
2. $i + 1 = f$
3. $f \leq 2^h$
4. $n + 1 \leq 2^{h+1}$

Question 6. Proposer une interprétation graphique de chacune des 4 formules précédentes.

Question 7. Montrer que pour tout $h \in \mathbb{N}$, il existe un arbre binaire strict pour lequel $n + 1 = 2^{h+1}$.

Soit $k \in \mathbb{N}^*$. On considère maintenant des arbres k-aires stricts, où chaque noeud interne a exactement k enfants.

Question 8. Proposer une généralisation des 4 formules montrées sur les arbres binaires stricts.

2 Parcours en profondeur

Un parcours d'arbre est un algorithme qui énumère les nœuds d'un arbre, un à un. Dans la suite, on identifiera les arbres et leur racine pour alléger les notations. Par exemple, on dira "les enfants de A" plutôt que "les arbres enracinés en les enfants de la racine de A".
Les algorithmes de parcours en profondeur ont le schéma suivant :

Algorithme 1 : Parcours(A)

Entrée(s) : A un arbre
si A ***est vide*** **alors**
  retourner
$G, D \leftarrow$ **enfants**(A);
Pré-traitement();
Parcours(G);
Parcours(D);
Post-traitement();

Les phases de pré/post-traitement dépendent de ce que l'on est en train de faire. Par exemple, écrivons une fonction qui affiche les étiquettes d'un arbre binaire :

```
let rec lister_pre (a: int arbre) : unit = match a with
  | Vide -> ()
  | Noeud(n, g, d) ->
    print_int n;
    lister_pre g;
    lister_pre d

let rec lister_post (a: int arbre) : unit = match a with
  | Vide -> ()
  | Noeud(n, g, d) ->
    lister_post g;
    lister_post d;
    print_int n
```

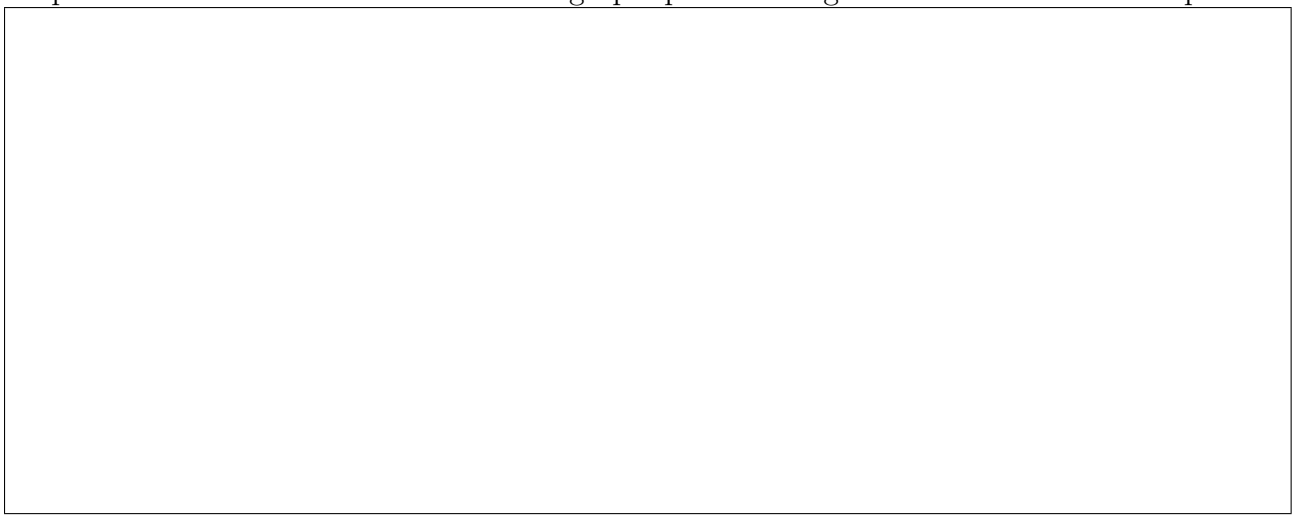
Voyons ce qu'affiche chacune des deux lorsqu'on les appelle sur un même arbre :

On parle de parcours ***préfixe*** lorsque l'on traite les nœuds avant leurs enfants (i.e. lorsque l'on fait un pré-traitement), et de parcours ***postfixe*** lorsque l'on traite les nœuds après leurs enfants (post-traitement).

On peut étendre la définition de ces deux parcours aux arbres d'arité quelconque. Pour les arbres binaires, il existe un troisième type de parcours en profondeur, dit ***infixe***, qui consiste à traiter chaque nœud **entre** ses deux enfants :

```
let rec lister_inf a = match a with
  | Vide -> ()
  | Noeud(n, g, d) ->
    lister_inf g;
    print_int n;
    lister_inf d
```

Ce parcours consiste en fait à lire l'arbre graphiquement de gauche à droite ! Par exemple :



Exemple 5. On considère un arbre binaire strict, étiqueté par $\{+, -, \times, /\} \cup \mathbb{N}$, tel que les feuilles sont étiquetées par $\mathbb{N}$, et les nœuds internes par les opérateurs. Un tel arbre représente une expression arithmétique : si on l'affiche selon les ordres infixes on obtient la notation infixe standard de l'expression :