Algorithmique et programmation fonctionnelle

Feuille 9 - Arbres planaires

Dans cette feuille nous introduisons les *arbres planaires*. Ce sont des arbres où il n'y a aucune contrainte sur l'arité des noeuds.

1 Définition

Informellement, un *arbre planaire* est un type d'arbre dans lequel il n'y a aucune limite sur le nombre de fils que peut avoir un nœud dans celui-ci. C'est-à-dire que si t est un arbre planaire, le nombre de fils d'un nœud x peut être n'importe quel entier positif : x peut avoir zéro fils (dans ce cas, x est une feuille), un fils, deux fils, trois fils, quatre fils, cinq fils, etc…En revanche, il y a un ordre sur les fils de x : il y a un premier fils, un second fils, un troisième fils, etc…On représente un *arbre planaire* dans la Figure 6.1.

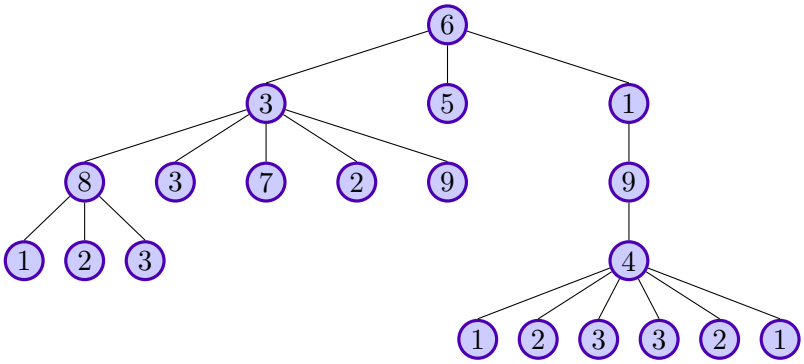


FIG. 6.1 : Un arbre planaire

Pour être plus précis, on va utiliser une définition récursive des arbres planaires.

Définition récursive des arbres planaires

Un ***arbre planaire*** est constitué de deux éléments :

1. Un ***nœud étiqueté*** par un objet de type quelconque qu'on appelle la ***racine*** de l'arbre planaire.
2. Une ***liste finie d'arbres planaires*** (potentiellement vide) qu'on appelle les ***fils*** de la racine.

Les arbres planaires sont toujours non-vides

Par définition, un arbre planaire contient toujours ***au moins un nœud*** (sa racine). Contrairement à ce que nous avons fait pour les arbres binaires, on ne considère donc pas d'arbre planaire "vide".

Pour parler des fils d'un noeud, on utilisera fréquemment la notion de ***forêt***.

Définition de forêts

Une ***forêt*** est une liste (possiblement vide) d'arbres planaires.

Par définition, les deux éléments composant un arbre planaire sont sa racine et une forêt contenant la liste des fils de celle-ci.
On donne un exemple de forêt non-vide dans la Figure 6.2.

Cas de base

Le cas de base de la définition récursive d'arbre planaire est le cas où la forêt des fils de la racine est ***vide***. Dans ce cas, l'arbre est composée d'un unique nœud (sa racine) qui est une feuille.

Exercice 1 : Squelettes d'arbre planaires

Dessiner tous les squelettes d'arbres planaires à 4 nœuds.

2 Implémentation des arbres planaires en OCaml

Puisque chaque arbre planaire est composé d'un nœud étiqueté et d'une liste de fils qui sont eux-mêmes des arbres planaires, on utilisera le type suivant pour implémenter des arbres planaires en OCaml

```
type 'a ptree = Pnode of 'a * 'a ptree list
```

Voici un exemple d'arbres planaires :

```
let t1 = Pnode (1, []);;
let t2 = Pnode (1, [Pnode (2, [])]);;
let t3 = Pnode (1, [Pnode (2,[Pnode (3, [])])]);;
let t4 = Pnode  (1, [Pnode (2, []); Pnode (3, [Pnode (4, [])] )]);;
```

Pour appliquer un traitement à la forêt des fils d'un noeud, on pourra utiliser les fonctions de la bibliothèque List d'OCaml, en particulier la fonction suivante :
`map : ('a -> 'b) -> 'a list -> 'b list` qui prend en paramètre une fonction `f : ('a -> 'b)` et une liste `[a_1; a_2; ...; a_n]` et qui renvoie la liste `[f a_1; f a_2; ...; f a_n]`.

Exercice 2 : Fonctions usuelles

1. Définir une variable `t_ex` qui représente l'arbre planaire donné dans la Figure 6.1.
2. Écrire une fonction `ptree_arity` qui prend en entrée un arbre planaire et retourne l'arité de sa racine.
3. Écrire une fonction `ptree_size` qui prend en entrée un arbre planaire et retourne sa taille.
4. Écrire une fonction `ptree_height` qui prend en entrée un arbre planaire et retourne sa hauteur.

3 Propriétés et codage par les arbres binaires

Exercice 3 : Dénombrement des arbres planaires

On dit que deux arbres planaires ont le *même squelette* si ils ont la même structure et ne diffèrent que par les étiquettes de leur nœuds. Pour tout entier $n \geqslant 1$, on note $f(n)$ le nombre de squelettes d'arbre planaires de taille n *distincts*.

1. Donnez les valeurs de $f(1)$, $f(2)$, $f(3)$ et $f(4)$.
2. Justifiez la relation suivante :

$$f(1) = 1 \qquad \text{et} \qquad f(n) = \sum_{i=1}^{n-1} (f(i) \times f(n-i)) \quad \text{pour } n \geqslant 2.$$

3. Combien vaut $f(n)$?

Il existe un *codage* des arbres planaires par des arbres binaires. Plus précisément, on peut coder n'importe quel arbre planaire par un arbre binaire *de même taille*.

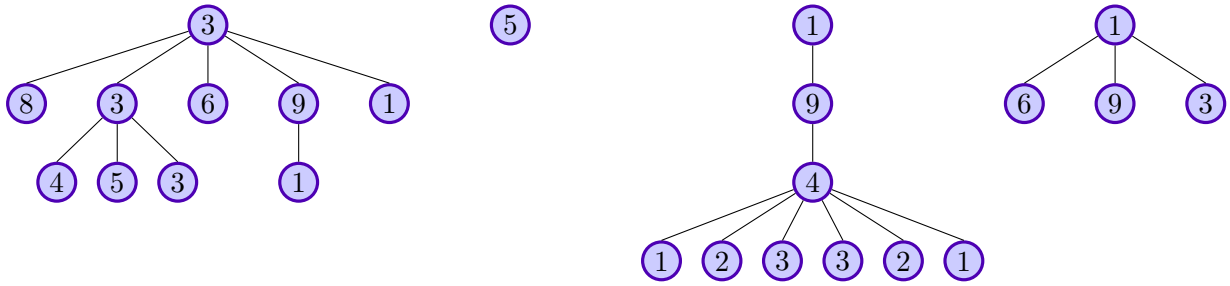
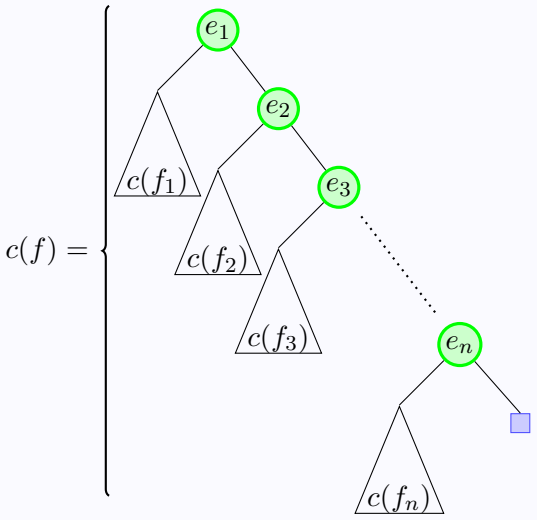


Fig. 6.2 : Une forêt contenant quatre arbres planaires

Exercice 4 : Codage fils-ainé/frère-droit

On va définir une fonction $c : f \mapsto c(f)$ qui associe un arbre binaire $c(f)$ à toute forêt d'arbres planaires f. La définition est récursive. D'abord, on considère le cas où f est une forêt vide (elle ne contient aucun arbre planaire). Dans ce cas, $c(f)$ est l'arbre binaire vide (noté $\square$ dans les figures). Sinon, f est une séquence non-vide $t_1, \ldots, t_n$ d'arbres planaires avec $n \geqslant 1$. Pour tout $i \leqslant n$, on note e_i l'étiquette de la racine de t_i et f_i la séquence de fils de cette racine (par définition f_i est une forêt). Par récursion, les arbres binaires $c(f_1), \ldots, c(f_n)$ associés aux forêts $f_1, \ldots, f_n$ dont bien définis. L'arbre binaire $c(f)$ associé à f est défini comme suit :



1. Dessinez l'image par c de la forêt de la Figure 6.2
2. La fonction $c : f \mapsto c(f)$ est-elle une bijection ? Justifiez votre réponse.
3. En utilisant la fonction "c" définir un codage des arbres planaires par les arbres binaires. Ce codage est-il toujours bijectif ? Quelle est la forme des arbres binaires produits par votre codage ?

Exercice 5 : Codage par un arbre binaire

1. Écrire une fonction `forest_to_btree : 'a ptree list -> 'a btree` qui prend en entrée une forêt f et retourne son codage $c(f)$ par un arbre binaire tel qu'il est défini dans l'éxercice précédent.
2. Écrire une fonction `ptree_to_btree : 'a ptree -> 'a btree` qui prend en entrée un arbre planaire et retourne son codage par un arbre binaire.

Exercice 6 : Décodage vers arbres planaires

1. Écrire une fonction `btree_to_forest : 'a btree -> 'a ptree list` qui prend un arbre binaire t en entrée et retourne la forêt codée par celui-ci.
2. Écrire une fonction `btree_to_ptree : 'a btree -> 'a ptree` qui prend un arbre binaire t en entrée et retourne l'arbre planaire par celui-ci si il est bien définit (dans le cas contraire, la fonction devra retourner un message d'erreur).

4 Parcours d'arbres planaires

Exercice 7 : Parcours prefixe d'un arbre planaire

Écrire une fonction `prefixe_dfs : 'a ptree -> 'a list` qui prend un arbre planaire et retourne la liste de ses étiquettes obtenue en utilisant un parcours préfixe.
On utilisera une pile (implémentée par une liste) de la même manière que pour le parcours préfixe des arbres binaires. Mais l'algorithme restera en $O(n^2)$ à cause de la concaténation de listes.

Nous allons maintenant proposer un algorithme de parcours suffixe des arbres planaires qui a une meilleure complexité que le parcours préfixe précédent.

Exercice 8 : Parcours suffixe d'un arbre planaire

On veut écrire un algorithme qui prend en entrée un arbre planaire t et retourne la liste $\ell(t)$ contenant les étiquettes des nœuds de t obtenue en effectuant un parcours *suffixe*. La définition du parcours suffixe pour les arbres planaires généralise celle sur les arbres binaires. Soit t un arbre planaire, e l'étiquette de sa racine et $t_1, \dots t_n$ la séquence de ses sous-arbres. La liste $\ell(t)$ des étiquettes des nœuds de t obtenue en effectuant un parcours *suffixe* est la suivante (on utilise le symbole "@" pour la concaténation de listes) ;

$$\ell(t) = \ell(t1) \mathbin{@} \ell(t_2) \mathbin{@} \cdots \mathbin{@} \ell(t_n) \mathbin{@} [e].$$

1. Écrire une fonction `suffixe_dfs0 : 'a ptree -> 'a list` qui prend un arbre planaire et retourne la liste de ses étiquettes obtenue en utilisant un parcours suffixe. Cette première fonction implémentera simplement la définition. Quelle est sa complexité ?

On veut maintenant écrire un algorithme plus efficace qui n'utilise *pas* la concaténation de listes. Pour cela, on va utiliser les forêts. Si f est une forêt contenant la séquence d'arbres planaires $t_1, \dots, t_n$, on note $\ell(f) = \ell(t1) \mathbin{@} \ell(t_2) \mathbin{@} \cdots \mathbin{@} \ell(t_n)$.

2. Pour toute forêt f, on note $r(f)$ le nombre de nœuds *racine* dans f et $d(f)$ le nombre de nœuds qui ne sont *pas une racine*.

   On considère une forêt f non-vide et la séquence $t_1, \dots, t_n$ d'arbres planaires qui la composent. Montrer que si t_1 contient *au moins* deux nœuds, alors il existe une forêt f' telle que $\ell(f) = \ell(f')$, $r(f') = r(f) + 1$ et $d(f') = d(f) - 1$.

3. En utilisant la question précédente, décrire un algorithme qui prend en entrée une forêt f et retourne $\ell(f)$ (sans utiliser la concaténation de listes). On justifiera que la procédure termine et on donnera sa complexité.

Exercice 9 : Parcours suffixe

Écrire une fonction `suffix_path : 'a ptree -> 'a list` qui prend un arbre planaire et retourne la liste de ses étiquettes obtenue en utilisant un parcours suffixe efficace.