COMPOSITION D'INFORMATIQUE (Option) n°1

Durée : 2h

⋆ ⋆ ⋆

Exercice 1

1. Écrire une fonction `minimum : 'a array -> 'a` qui renvoie l'élément minimum dans un tableau. On considèrera qu'on ne fera d'appel à la fonction que pour des tableaux non vides et on ne demande pas de vérifier cette condition.
2. Écrire une fonction `sous_chaine : string -> int -> int -> string` telle que si on pose $s = c_0c_1 \dots c_{n-1}$, $i \in \mathbb{N}$ et $j \in \mathbb{N}$, alors `sous_chaine s i j` renvoie la sous-chaîne de caractères $c_ic_{i+1} \dots c_{j-2}c_{j-1}$. On considèrera que $n > 0$ et que $i, j \in [\![0, n]\!]$, $i < j$ et on ne demande pas de vérifier ces conditions.

Exercice 2

1. On donne la fonction suivante :

```
let rec mystere lst n = match lst, n with
   | [], _ -> [], []
   | _, 0 -> [], lst
   | x :: q, _ -> let lst1, lst2 = mystere q (n - 1) in x :: lst1, lst2;;
```

   Déterminer la signature de la fonction `mystere` et donner une description de ce qu'elle fait.
2. Écrire une fonction `avant_dernier : 'a list -> 'a` qui renvoie l'avant-dernier élément d'une liste. On écrira un message d'erreur si nécessaire.

Problème

On définit les *arbres* (binaires entiers) de la façon suivante :

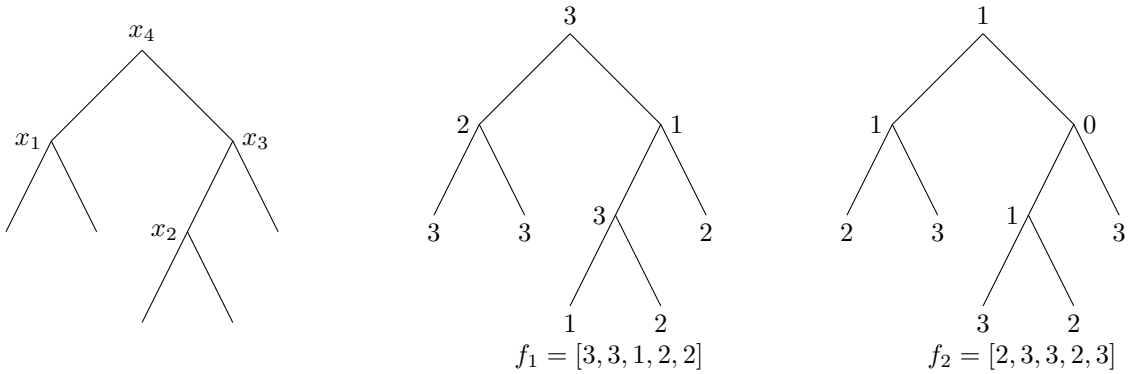
- Une *feuille* est un arbre.
- Si g et d sont deux arbres, le couple (g, d) est un arbre.

Un arbre de la forme (g, d) est un *nœud interne*, dont g est le *fils gauche* et d le *fils droit*. Par la suite, on note A_n l'ensemble des arbres possédant n feuilles ($n \geq 1$). Les feuilles d'un arbre de A_n sont numérotées de 1 à n dans l'ordre d'un parcours postfixe (dit aussi en profondeur d'abord) et de gauche à droite.

Un *flot* f de taille n est une suite de n entiers égaux à 1, 2 ou 3. Pour un arbre a de A_n fixé, cette suite définit une fonction des sous-arbres de a dans les entiers, également notée f.
La fonction f associe le i-ème entier du flot f à la i-ème feuille de l'arbre a, et elle s'étend récursivement aux nœuds internes, par la formule :

$$f((g, d)) = f(g) + f(d) \text{ (modulo 4)},$$

c'est-à-dire que si $a = (g, d)$, alors $f(a)$ est le reste de la division euclidienne par 4 de la somme $f(g) + f(d)$.
On note F_n l'ensemble des flots de taille n. Un flot f de F_n est dit *compatible* avec un arbre a de A_n si, pour tout nœud interne x de a, on a : $f(x) \neq 0$. La figure suivante représente un arbre a à 5 feuilles (dont les nœuds internes sont désignés par $x_1, \dots, x_4$), et deux flots. On notera que le premier flot est compatible avec a, tandis que le second ne l'est pas, car $f_2(x_3) = 0$.



Les arbres sont représentés en machine par la structure de donnée usuelle :

```
type arbre =
   | Feuille
   | Interne of arbre * arbre;;
```

1. Écrire une fonction `compte_feuilles : arbre -> n` qui prend un arbre en argument et renvoie le nombre de ses feuilles.
2. Dans cette question, a désigne un arbre de A_n et f un flot de F_n.
   (a) Montrer que a possède $n-1$ nœuds internes.
   (b) On suppose que a n'est pas réduit à une feuille et on écrit $a = (g, d)$. Soit f un flot *compatible* avec a. En fonction des valeurs possibles de $f(a)$, donner les valeurs possibles que peut prendre le couple $(f(g), f(d))$.
   (c) On pose $F(a, v)$ le nombre de flots compatibles avec a tels que $f(a) = v$, pour $v \in \{1, 2, 3\}$. Montrer par récurrence que $F(a, v)$ ne dépend que de a et que $F(a, v) = 2^{n-1}$. En déduire le nombre de flots compatibles avec a.
3. Un flot sera représenté par le type suivant :

   ```
   type flot = int array;;
   ```

   On cherche à écrire une fonction `compatible : flot -> arbre -> bool` qui teste la compatibilité d'un flot avec un arbre. On complètera, avec des explications, la fonction écrite ci-dessous (qu'on ne demande pas de réécrire complètement) :

   ```
   let compatible f a =
      let num_feuille = ref (-1) in
      let b = ref true in
      let rec image_par_f = function
         | Feuille -> incr num_feuille; f.(!num_feuille)
         | Interne(g, d) -> (* Partie à compléter *)
      in let fa = image_par_f a in
      !b;;
   ```

4. On représente maintenant un flot de la manière suivante :

   ```
   type flot = int list;;
   ```

   (difficile) Écrire une fonction `compatible : flot -> arbre -> bool` qui teste la compatibilité d'un flot avec un arbre. On interdira l'usage de références et de boucles dans cette fonction.
5. On pose $C_n = \mathrm{Card}(A_n)$ le nombre d'arbres à n feuilles.
   (a) Déterminer une formule de récurrence permettant de calculer C_n en faisant intervenir tous les C_k, pour $k \in [\![1; n-1]\!]$.
   (b) En déduire une fonction `catalan : int -> int array` telle que `catalan n` renvoie un tableau `t` de taille n tel que `c.(i)` vaut C_i pour tout $i \in [\![1, n-1]\!]$.
   (c) Déterminer le nombre de multiplications effectuées par la fonction précédente.
6. *(très difficile)* Écrire une fonction `genere_arbre : int -> int -> arbre` qui prend en argument un entier n et un nombre m compris entre 0 et $C_n - 1$ et renvoie un arbre de A_n. Deux entiers différents doivent renvoyer un arbre différent. On supposera pour cette question que le tableau donné par la fonction `catalan` a été enregistré dans une variable globale `c` de taille suffisante et ne sera plus modifié.
   Indication : On commencera par réfléchir à la numérotation des arbres de A_n et on considèrera une partition adéquate de $[\![0, C_n - 1]\!]$.

⋆ ⋆ ⋆