Persistance additive et multiplicative

Calculatrices interdites. Pensez à numéroter vos feuilles.
Si vous repérez une erreur d'énoncé, signalez le sur votre copie et continuez votre composition.
Les questions doivent apparaître dans l'ordre du sujet.

Rappel : pour manipuler des listes chaînées en OCaml, il est naturel d'utiliser des fonctions récursives et non des fonctions itératives. Si c'est pertinent vous pouvez écrire des fonctions itératives, mais dans la plupart des questions l'utilisation de fonctions itératives à la place de fonctions récursives sera sanctionné.

Rappel : attention de ne pas confondre les notions d'« entier » et de « chiffre ». Un entier est un élément de $\mathbb{N}$, un chiffre est un élément de $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Soit $n \in \mathbb{N}$ un entier naturel. On s'intéresse aux opérations consistant à additionner ou à multiplier les chiffres de n entre eux. Par exemple, lorsqu'on multiplie les chiffres du nombre $15\,638$, on obtient $1 \times 5 \times 6 \times 3 \times 8 = 720$ et lorsqu'on multiplie les chiffres de 720 on obtient $7 \times 2 \times 0 = 0$. La "persistance" d'un entier est le nombre d'itérations nécessaires pour obtenir un résultat avec un seul chiffre.

1 Liste des chiffres d'un entier

Dans tout le sujet, étant donné un entier $n \in \mathbb{N}$, on note `L(n)` la liste contenant les chiffres de n. En particulier, si $n \neq 0$ alors le premier élément de `L(n)` n'est pas `0`. Par exemple :

n	0	5	1560380
L(n)	[0]	[5]	[1;5;6;0;3;8;0]

1. Écrire une fonction `rev: int list -> int list` qui inverse l'ordre des éléments de la liste donnée en entrée. Par exemple, `rev [1;2;3;4]` s'évalue en `[4;3;2;1]`.

En OCaml, la fonction « `for_all: ('a -> bool) -> 'a list -> bool` » du module `List` prend en entrée une fonction « `pred: 'a -> bool` » ainsi qu'une liste « `li: 'a list` ». Elle renvoie `true` si `(pred e)` s'évalue en `true` pour tout élément `e` de `li` ; et renvoie `false` sinon.

2. À l'aide de la fonction `List.for_all`, écrire une fonction `verif: int list -> bool` qui renvoie `true` si tous les éléments de la liste donnée en entrée appartiennent à $[\![0; 9]\!]$; et `false` sinon.

3. (a) Écrire une fonction `intToList: int -> int list` qui prend en entrée un entier `n` et renvoie `L(n)`. Si `n < 0`, votre fonction déclenchera une erreur. La fonction obtenue devra être de complexité linéaire en la taille de la liste en sortie.
   (b) Écrire une fonction `listToInt: int list -> int` qui prend en entrée une liste `li` et renvoie l'entier `n` vérifiant `li = L(n)`. Si aucun entier $n \in \mathbb{N}$ ne vérifie cette égalité, votre fonction déclenchera une erreur. La fonction obtenue devra être de complexité linéaire en la taille de la liste en entrée.

4. Donner en la justifiant la taille de `L(n)` en fonction de `n`. On pourra utiliser le logarithme en base 10 :

$$\log_{10} : n \mapsto \frac{\ln(n)}{\ln(10)}$$

2 Persistance additive

Dans cette partie, $S : \mathbb{N} \to \mathbb{N}$ est la fonction qui à un entier associe la somme de ses chiffres. Soit $n \in \mathbb{N}$ un entier fixé. On note $(u_i)_{i\in\mathbb{N}}$ la suite définie par :

$$\begin{cases} u_0 = n. \\ u_{i+1} = S(u_i) \text{ pour tout } i \geqslant 0. \end{cases}$$

Par exemple avec n = 9 897 787 on obtient :

$$u_0 = 9\,897\,787, \qquad u_1 = 55, \qquad u_2 = 10, \qquad u_3 = 1, \qquad u_4 = 1, \qquad u_5 = 1, \qquad \ldots$$

On appelle ***persistance additive*** de n le plus petit entier $k \in \mathbb{N}$ tel que u_k est un nombre avec un seul chiffre. Par exemple, la persistance additive de 9 897 787 est 3.

5. (a) Écrire une fonction `somme: int list -> int` qui renvoie la somme des éléments d'une liste.
   (b) En déduire une fonction récursive `persAdd: int -> int` qui prend en entrée un entier n et renvoie la persistance additive de n. Si n < 0, votre fonction déclenchera une erreur.
6. (a) Montrer que pour tout $k \in \mathbb{N}$, il existe un entier $n \in \mathbb{N}$ dont la persistance additive est k.

   (b) Écrire une fonction sans argument `exPersAdd4: unit -> int list` qui renvoie une liste L(n) où $n \in \mathbb{N}$ est un entier dont la persistance additive est égale à k = 4. On expliquera succinctement la procédure utilisée.

3 Persistance multiplicative

3.1 Définition

Soit $P : \mathbb{N} \to \mathbb{N}$ la fonction qui à un entier associe le produit de ses chiffres. Soit $n \in \mathbb{N}$ un entier fixé et $(v_i)_{i\in\mathbb{N}}$ la suite définie par :

$$\begin{cases} v_0 = n. \\ v_{i+1} = P(v_i) \text{ pour tout } i \geqslant 0. \end{cases}$$

La ***persistance multiplicative*** de n est le plus petit entier $k \in \mathbb{N}$ tel que v_k est un nombre avec un seul chiffre. Dans la question 6a, on a montré que pour tout $k \in \mathbb{N}$ il existe un entier dont la persistance additive est k. Dans le cas de la persistance multiplicative, il est conjecturé (mais personne n'a réussi à le démontrer) que la persistance multiplicative maximale possible est 11.

7. Quelle est la persistance multiplicative de 678 ? Justifier.
8. Montrer que pour tout $n \in \mathbb{N}$, si n est un nombre avec au moins deux chiffres alors P(n) < n.

En OCaml, la fonction « `iter: ('a -> unit) -> 'a list -> unit` » du module `List` prend en entrée une fonction « `f: 'a -> unit` » ainsi qu'une liste $[a_1; a_2; \ldots; a_k]$ et exécute la suite d'instructions :

$$f\ a_1; f\ a_2; \ldots; f\ a_k;$$

9. Compléter le code ci-dessus pour que la fonction `prodChiffres` renvoie le produit des chiffres de n.

```
let prodChiffres (n: int): int =
  let res = ... in
  let f e =
    ...
  in
  List.iter f (intToList n);
  ...;;
```

Soit $n_0 \in \mathbb{N}$. Afin de calculer la persistance multiplicative de tous les entiers $n \in [\![0; n_0]\!]$, on va construire un tableau `tabPM` tel que `tabPM.(n)` contienne la persistance multiplicative de `n`.

10. (a) Expliquer comment construire le tableau `tabPM` en faisant en sorte que le nombre d'appels à la fonction `prodChiffres` soit en $\mathcal{O}(n_0)$.
    (b) En déduire une fonction `makeTabPM: int -> int array` qui prend en entrée n_0 et renvoie `tabPM`.
    (c) Donner le nombre exact d'appels à la fonction `prodChiffres` dans la fonction `makeTabPM`.

3.2 Plus petit entier avec une persistance multiplicative donnée

Pour tout $k \in \mathbb{N}$, on note N_k le plus petit entier naturel dont la persistance multiplicative vaut k. Par exemple, $N_0 = 0$ et $N_1 = 10$.

11. Déterminer N_2 et N_3. Justifier votre réponse.

On souhaite maintenant écrire une fonction qui prend en entrée un entier k et renvoie N_k. Pour cela, on commence par déterminer les persistances multiplicatives de tous les entiers inférieurs ou égaux à $n_0 = 10$ à l'aide de la fonction `makeTabPM`. Si l'un de ces entiers a une persistance multiplicative égale à k, on renvoie N_k ; sinon on recommence avec $n_0 = 10^2$, puis $n_0 = 10^3$, puis $n_0 = 10^4$, ... jusqu'à obtenir le résultat voulu.

12. Écrire une fonction `getNk: int -> int` qui prend en entrée $k \geqslant 0$ et renvoie N_k en utilisant la procédure décrite ci-dessus.

L'exécution de la fonction `getNk` permet d'établir le tableau suivant :

k	4	5	6	7	8	9
N_k	77	679	6 788	68 889	2 677 889	26 888 999

Malheureusement, `getNk` est trop lente pour obtenir les valeurs de N_k lorsque $k \geqslant 10$.

3.3 Persistances multiplicatives 10 et 11

13. Soit $k \geqslant 3$. Montrer que :
    (a) $L(N_k)$ ne contient ni de 0, ni de 1.
    (b) $L(N_k)$ est triée par ordre croissant.
    (c) $L(N_k)$ ne peut pas contenir plusieurs fois le chiffre 2. Idem pour 3, 4 et 6.
    (d) $L(N_k)$ ne peut pas contenir à la fois un chiffre pair et 5.

Dans la suite pour $c \in [\![0; 9]\!]$, on utilisera la notation $c^\star$ pour indiquer qu'il existe $\ell \in \mathbb{N}$ tel que le chiffre c se répète ℓ fois. Par exemple, on dira que « n_1 est de la forme $29^\star$ » si le chiffre de poids fort de n_1 est 2 et que tous ses autres chiffres sont des 9 ; c'est à dire si :

$$n_1 \in \{2; 29; 299; 2999; 29999; \ldots\}$$

De même, « n_2 est de la forme 9* » et « n_3 est de la forme 57*9* » signifient que :

$$\begin{aligned} n_2 \in &\{9; 99; 999; 9999; \ldots\} \\ n_3 \in &\{5; 57; 577; 5777; 57777; \ldots\} \cup \\ &\{59; 579; 5779; 57779; 577779; \ldots\} \cup \\ &\{599; 5799; 57799; 577799; 5777799; \ldots\} \cup \ldots \end{aligned}$$

14. Soit $k \geqslant 3$. Montrer que si le chiffre de poids fort de N_k est un 2 alors N_k est de la forme 267*8*9* ou 27*8*9*.

15. (a) Écrire une fonction `ajout : int -> int list list -> int list list` qui prend en entrée un entier `c` ainsi qu'une liste de listes `li` et ajoute `c` au début de chacune des sous-listes. Par exemple :

```
(* ajout 2 [[7;8;9]; [3;7;7;8]; [5;7;9;9]] *)
[[2; 7; 8; 9]; [2; 3; 7; 7; 8]; [2; 5; 7; 9; 9]]
```

(b) Écrire une fonction `generer: int -> int list -> int list list` qui prend en entrée un entier `t` ainsi qu'une liste $[c_1; c_2; \ldots; c_k]$ et renvoie une liste contenant toutes les listes de taille `t` de la forme :

$$li_1 \mathbin{@} li_2 \mathbin{@} \ldots \mathbin{@} li_k$$

où la liste li_j ne contient que des éléments égaux à c_j. Par exemple, `(generer 4 [7; 8; 9])` contient les listes `L(n)` pour tous les $n \in \mathbb{N}$ de la forme 7*8*9* avec 4 chiffres :

```
(* generer 4 [7; 8; 9] *)
[[7; 7; 7; 7]; [7; 7; 7; 8]; [7; 7; 7; 9]; [7; 7; 8; 8]; [7; 7; 8; 9];
 [7; 7; 9; 9]; [7; 8; 8; 8]; [7; 8; 8; 9]; [7; 8; 9; 9]; [7; 9; 9; 9];
 [8; 8; 8; 8]; [8; 8; 8; 9]; [8; 8; 9; 9]; [8; 9; 9; 9]; [9; 9; 9; 9]]
```

Soit $k \geqslant 3$. En utilisant le même raisonnement que dans la question 14, il est possible de montrer que N_k est nécessairement de l'une des neuf formes suivantes :

267*8*9*	27*8*9*	347*8*9*	357*9*	37*8*9*
47*8*9*	57*9*	67*8*9*	7*8*9*	

Afin de calculer N_k, on génère donc tous les entiers de cette forme jusqu'à trouver N_k.

16. Écrire une fonction `getNkBis: int -> int` qui prend en entrée $k \geqslant 0$ et renvoie N_k en utilisant la procédure décrite ci-dessus. Votre fonction déclenchera une erreur lorsque $k \notin [\![0; 11]\!]$ (personne n'a jamais trouvé de nombre dont la persistance multiplicative est supérieure ou égale à 12).