Perles de Dijkstra

Consignes :

→ Calculatrices interdites.

→ Sur votre copie, numérotez vos feuilles et faites apparaître les questions dans l'ordre du sujet.

→ Si vous repérez une erreur d'énoncé, signalez le sur votre copie et continuez votre composition.

→ Si vous écrivez un programme qui n'est pas optimal en terme de complexité, vous n'aurez pas tous les points.

1 Préliminaires

Le but de cette partie est de réécrire les fonctions `length`, `(@)` et `filter` du module `List` d'OCaml.

1. Cette question doit être traitée sans recourir aux fonctions du module `List`.
   (a) Écrire une fonction `length: 'a list -> int` qui renvoie la taille de la liste `li` en entrée.
   (b) Écrire une fonction `concat: 'a list -> 'a list -> 'a list` qui prend en entrée deux listes `li1, li2` et renvoie `li1 @ li2`. Rappel : vous n'avez le droit d'utiliser `@`.
   (c) Écrire une fonction `filter: 'a list -> ('a -> bool) -> 'a list` qui prend en entrée une liste « `li: 'a list` », ainsi que « `pred: 'a -> bool` », et renvoie la liste composée de tous les éléments `e` de `li` tels que « `pred e` » s'évalue en `true`.

Dans la suite, on pourra utiliser « `length` », « `List.length` », « `concat` », « `@` », « `filter` » et « `List.filter` ». Toutefois, n'utilisez pas ces fonctions si cela détériore le temps d'exécution de votre programme. En particulier, utiliser `List.length` à chaque appel d'une fonction récursive est rarement optimal en terme de complexité.

2 Énoncé du problème

Énoncé informel. Dans le "problème des perles de Dijkstra", on dispose de perles bleues, blanches et rouges qu'on souhaite placer sur un fil pour former un collier en respectant certaines contraintes. Pour être une solution du problème, un collier ne doit pas contenir deux séquences consécutives identiques.

Formalisation. Dans tout l'énoncé, une perle bleue sera représentée par l'entier 0, une perle blanche par 1 et une perle rouge par 2. Un collier correspond à une suite finie d'éléments de $\{0, 1, 2\}$. Par exemple, le collier 01102022 est composé de huit perles dont trois sont bleues, deux sont blanches et 3 sont rouges. La notation ε désigne l'unique collier de taille 0.

Si c_1 et c_2 sont deux colliers, on note $c_1 \cdot c_2$ la concaténation de c_1 et c_2. Par exemple, si $c_1 = 112002$ et $c_2 = 2202$, alors $c_1 \cdot c_2 = 1120022202$. Pour c_1 et c_2 deux colliers, on dit que :

→ c_1 est un ***préfixe*** de c_2 s'il existe un collier c_3 tel que $c_2 = c_1 \cdot c_3$. Par exemple, 1120 est un préfixe de 1120120.

→ c_1 est un ***suffixe*** de c_2 s'il existe un collier c_3 tel que $c_2 = c_3 \cdot c_1$. Par exemple, 120 est un suffixe de 1120120.

→ c_1 est un ***facteur*** de c_2 s'il existe deux colliers c_3 et c_4 tels que $c_2 = c_3 \cdot c_1 \cdot c_4$. Par exemple, 1201 et 01 sont des facteurs de 1120120.

Un collier c_0 sera qualifié de "collier solution" s'il respecte la contrainte suivante :

Pour tout $c \neq \varepsilon$, le collier $c \cdot c$ n'est pas un facteur de c_0.

Par exemple, $c_0 = 12010121012$ n'est pas collier solution. En effet, si $c = 01$, alors $c \cdot c = 0101$ est un facteur de c_0.

Soit $n \in \mathbb{N}$. L'objectif du sujet est de construire des colliers solutions de taille n. Par exemple :

→ Pour $n = 0$, un seul collier est solution : le collier ε.

→ Pour $n = 1$, trois colliers sont solutions : 0, 1 et 2.

→ Pour $n = 2$, six colliers sont solutions : 01, 02, 10, 12, 20 et 21.

2. Soit c_0 un collier.
   (a) Montrer que ε est un préfixe et un suffixe de c_0.
   (b) Montrer que c_0 est un préfixe et un suffixe de lui-même.
   (c) Soit c un préfixe ou un suffixe de c_0. Montrer que c est un facteur de c_0.

3. (a) Le collier $c_0 = 1202112021$ est-il un collier solution ? Justifier.
   (b) Donner sans justification tous les colliers solutions de taille 3.
   (c) Déterminer tous les colliers solutions de taille 8 dont $c = 102010$ est un préfixe. On expliquera succinctement la démarche utilisée.
   (d) Montrer qu'il n'existe pas de collier solution de taille $n \geqslant 12$ dont $c = 01021201021$ est un suffixe.

3 Construction de colliers solutions en OCaml

4. Écrire une fonction `verif012: int list -> bool` qui renvoie `true` si tous les éléments de la liste en entrée sont dans $\{0, 1, 2\}$, et `false` sinon.

Dans la suite, un collier sera représenté par une liste d'entiers :

```
(* Une liste de type collier ne doit contenir que des 0, 1, 2 *)
type collier = int list;;
```

Lorsqu'une fonction prend en entrée une liste `c` de type `collier`, on pourra supposer sans le vérifier que tous les éléments de `c` sont dans $\{0, 1, 2\}$.

5. (a) Écrire une fonction `estPref: collier -> collier -> bool` qui prend en entrée deux colliers c_1 et c_2, et indique si c_1 est un préfixe de c_2.
   (b) Écrire une fonction `couper: collier -> int -> (collier * collier)` qui prend en entrée un collier `c` ainsi qu'un entier `n` $\in \mathbb{N}$ et renvoie deux collier `c1, c2` où `c1` est le préfixe de taille `n` de `c` et `c = c1 @ c2`. Si `n` est trop grand par rapport à la taille de `c`, votre fonction déclenchera une erreur. Par exemple, pour $c = 12010212010$ et $n = 4$, on obtient $c_1 = 1201$ et $c_2 = 0212010$.
   (c) À l'aide des deux fonctions précédentes, écrire une fonction

   `doubleFact: collier -> int -> int -> bool`

   qui prend en entrée un collier c, ainsi que la taille de c et un entier $n \in \mathbb{N}$, et renvoie `true` si et seulement si c admet un facteur de la forme $c_1 \cdot c_1$ où c_1 est un collier de taille n. Attention : votre fonction ne doit jamais déclencher d'erreur, en particulier lors des appels à `couper`.

6. À l'aide d'une boucle `while` et des fonctions précédentes, écrire une fonction

   `estSol: collier -> bool`

   qui indique si le collier en entrée est un collier solution. La boucle `while` devra s'arrêter dès que possible.

7. (a) Écrire une fonction `ajout: collier list -> collier list` qui pour chaque collier c de la liste en entrée créé trois colliers : $0 \cdot c$, $1 \cdot c$ et $2 \cdot c$. Par exemple, « `ajout [[0;1]; [0]; []]` » vaut :

   `[[0;0;1]; [1;0;1]; [2;0;1]; [0;0]; [1;0]; [2;0]; [0]; [1]; [2]]`

   (b) Écrire une fonction `tousCol: int -> collier list` qui prend en entrée un entier $n \in \mathbb{N}$ et renvoie la liste de tous les colliers de taille n (les colliers peuvent être solutions ou non).
   (c) Écrire une fonction `tousColSol: int -> collier list` qui prend en entrée un entier $n \in \mathbb{N}$ et renvoie la liste de tous les colliers solutions de taille n.

4 Mots de Thue-Morse

Dans cette partie, vous aurez besoin d'utiliser la fonction « `String.init: int -> (int -> char) -> string` » qui prend en entrée un entier `n` $\in \mathbb{N}$ ainsi que « `f: int -> char` » et renvoie la chaîne de caractères `s` de taille `n` telle que pour tout `i` $\in [\![$`0, n - 1`$]\!]$, `s.[i] = f i`.

4.1 Définitions

On appelle ***mot*** une chaînes de caractères contenant des a et des b :

```
(* Une chaîne de caractères de type mot ne doit contenir que des a et b *)
type mot = string;;
```

Lors de l'écriture d'un mot, on omettra les guillemets, par exemple le mot $baabb$ correspond à la chaîne de caractères `"baabb"`. Comme dans le cas des colliers, on note $s_1 \cdot s_2$ la concaténation de deux mots s_1, s_2 et on introduit les notions de préfixes, suffixes, facteurs (ce sont les mêmes définitions que pour les colliers).

Soit ϕ la fonction qui à tout mot s associe le mot s' où chaque a est remplacé par ab et chaque b est remplacé par ba. Par exemple, $\phi(baabb) = baababbaba$. Pour tout $n \in \mathbb{N}$, on appelle ***mot de Thue-Morse*** et on note s_n le mot défini par :

$$\begin{cases} s_0 = a \\ \forall n \geqslant 0 : s_{n+1} = \phi(s_n) \end{cases}$$

Par exemple :

$$s_0 = a \qquad s_1 = ab \qquad s_2 = abba \qquad s_3 = abbabaab \qquad s_4 = abbabaabbaababba$$

8. Écrire une fonction `motTM: int -> string` qui prend en entrée un entier $n \in \mathbb{N}$ et renvoie s_n. Dans cette question, on demande de construire s_0, puis s_1, puis s_2, …, puis s_n. À chaque étape, vous devez utiliser la fonction `String.init` pour construire s_{i+1} à partie de s_i.

4.2 Construction d'un collier solution

L'objectif des questions suivantes est d'obtenir un collier solution pour le problème des perles de Dijkstra à partir des mots de Thue-Morse. Soit `mot_of_collier` la fonction suivante :

```
let rec mot_of_collier (c: collier): mot = match c with
  | [] -> ""
  | 0 :: q -> "a" ^ mot_of_collier q
  | 1 :: q -> "ab" ^ mot_of_collier q
  | 2 :: q -> "abb" ^ mot_of_collier q
  | _ -> failwith "mot_of_collier: le collier est invalide";;
```

9. Donner sans justification le valeur de « `mot_of_collier [1; 0; 1; 2; 0; 0]` ».
10. Soit $n \in \mathbb{N}$.
    (a) Montrer que s_n commence par un a.
    (b) Déterminer le nombre de caractères dans s_n. Montrer votre formule.
    (c) Montrer qu'il est impossible de trouver trois b consécutifs dans s_n.
    (d) Montrer qu'il existe un unique « `c: collier` » tel que « `mot_of_collier c` » s'évalue en s_n.
    (e) Écrire une fonction `collierTM: int -> collier` qui prend en entrée l'entier n et renvoie le collier `c` trouvé à la question 10d

Remarque. Le collier construit dans la question 10e est un collier de solution, mais il n'est pas demandé de le démontrer.

4.3 Autres constructions des mots de Thue-Morse

Le but de cette partie est de programmer deux autres méthodes pour construire les mots de Thue-Morse.

Méthode 2. Soit $i \in \mathbb{N}$ un entier. On note $\alpha(i)$ la somme des bits dans l'écriture en base 2 de i. Par exemple :

$$13 = (1101)_2 \qquad \text{donc} \qquad \alpha(13) = 1 + 1 + 0 + 1 = 3$$

11. (a) Pour tout $n \in \mathbb{N}$ et tout $i \in [\![0, 2^n - 1]\!]$, on note $s_{n,i}$ le caractère d'indice i de s_n. Montrer que $s_{n,i}$ est égal à a si $\alpha(i)$ est pair et à b si $\alpha(i)$ est impair.
    (b) En utilisant le résultat de la question précédente, écrire une fonction `motTM_bis: int -> string` qui prend en entrée un entier n et renvoie s_n.

Méthode 3. Pour tout $n \in \mathbb{N}$, on note u_n et v_n les deux mots définis par :

$$\begin{cases} u_0 = 0 \\ v_0 = 1 \end{cases} \qquad \forall n \in \mathbb{N} : \begin{cases} u_{n+1} = u_n \cdot v_n \\ v_{n+1} = v_n \cdot u_n \end{cases}$$

12. (a) Montrer que pour tout $n \in \mathbb{N}$, $u_n = s_n$.
    (b) En utilisant le résultat de la question précédente, écrire une fonction `motTM_ter: int -> string` qui prend en entrée un entier n et renvoie s_n.