DS3 d'informatique MPSI
20 mai 2011

Devoir surveillé

Durée : 2 heures

Aucun document n'est autorisé. **On pourra définir des fonctions non demandées explicitement, si cela facilite la programmation. Inutile de prouver la correction et la terminaison des fonctions écrites, sauf si on le demande explicitement. Les fautes de syntaxe seront sanctionnées.**

Identification de facteurs répétés dans un mot

On s'intéresse à l'identification de facteurs répétés dans un mot formé sur l'alphabet $\mathcal{A} = \{a, b, c, \ldots, z\}$. Soit $m = x_0 x_1 \ldots x_{n-1}$ un mot de longueur n formé sur $\mathcal{A}$ ($x_i \in \mathcal{A}$, pour tout $i \in [\![0, n-1]\!]$). Un mot sera représenté en Caml par une chaîne de caractères (type string).

Nous illustrerons dans la suite les différents algorithmes en prenant pour exemple le mot abracadabra.

```
# let m = "abracadabra";;
m : string = "abracadabra"
```

Rappels sur le type string : étant donné une chaîne de caractères s, dont les lettres (ou caractères) sont indicées de 0 à $n-1$, la commande `s.[i]` renvoie la lettre de s d'indice i. De plus, on dispose des fonctions string_length, sub_string.

Exemples :

```
# string_length m;;
- : int = 11

# for i = 0 to (string_length m - 1) do print_char m.[i]; print_char ' ' done;;
a b r a c a d a b r a - : unit = ()

(* sub_string chaîne indice_initial longueur *)
# sub_string m 2 5;;
- : string = "racad"
```

Nous définissons également une fonction numero_lettre permettant d'indicer les éléments de $\mathcal{A}$ par les entiers de 0 à 25 selon l'ordre usuel :

```
# let numero_lettre c = int_of_char c - 97;;
numero_lettre : char -> int = <fun>

# numero_lettre 'a';;
- : int = 0

# numero_lettre 'z';;
- : int = 25
```

Soit $k \in [\![1, n]\!]$ et $i, j \in [\![0, n-k]\!]$. Les rangs i et j sont dits *k-équivalents* si

$$x_i x_{i+1} \ldots x_{i+k-1} = x_j x_{j+1} \ldots x_{j+k-1},$$

c'est-à-dire si les facteurs de longueur k de m commençant aux indices i et j sont identiques.

Ce fait [1] est noté $< i, E_k, j >$.

Pour chaque $k \in [\![0, n]\!]$, E_k définit une relation d'équivalence (*i.e.* réflexive, symétrique, transitive) sur $[\![0, n-k]\!]$.

Cette relation d'équivalence E_k permet de regrouper les éléments de $[\![0, n-k]\!]$ par classes d'équivalences : la classe d'équivalence X_i de $i \in [\![0, n-k]\!]$ pour E_k est

$$X_i = \{j \in [\![0, n-k]\!], \quad < j, E_k, i > .$$

1. par souci de légèreté, nous n'avons pas fait figurer m dans cette notation, alors que ce fait dépend évidemment de m.

Les classes d'équivalences de $[\![0, n-k]\!]$ pour E_k sont les parties X de $[\![0, n-k]\!]$ pour lesquelles $X = X_i$ pour un certain $i \in [\![0, n-k]\!]$.

Exemple : pour $k = 1$, on a $X_0 = \{0, 3, 5, 7, 10\} (= X_3 = X_5 = X_7 = X_{10})$, $X_1 = \{1, 8\} (= X_8)$, $X_2 = \{2, 9\} (= X_9)$, $X_4 = \{4\}$ et $X_6 = \{6\}$.

Pour $k = 2$, on a par exemple $X_0 = \{0, 7\}$.

Nous numéroterons les r différentes classes d'équivalence pour E_k par les entiers de 0 à $r-1$, en respectant l'ordre d'apparition de ces classes lorsqu'on parcourt m de gauche à droite. À chaque $i \in [\![0, n-k]\!]$ nous associerons l'entier correspondant à sa classe d'équivalence. Nous pouvons ainsi obtenir un vecteur des classes d'équivalence de $[\![0, n-k]\!]$ pour E_k.

Exemple : pour $k = 1$, nous obtenons [|0; 1; 2; 0; 3; 0; 4; 0; 1; 2; 0|].

Pour $k = 2$, nous obtenons [|0; 1; 2; 3; 4; 5; 6; 0; 1; 2|].

Le but de ce problème est de construire le tableau des classes d'équivalences pour E_k (et pour un mot fixé).

Partie A – Préliminaires sur les vecteurs et piles

A.1 Définir une fonction map_vect : ('a -> 'b) -> 'a vect -> 'b vect, envoyant une fonction f et un vecteur t sur le vecteur des images des termes de t par f :

```
map_vect f [|a_0; ...; a_n|] = [|f a_0; ...; f a_n|]
```

A.2 Définir une fonction iter_vect : (int -> 'a) -> int -> int -> 'a vect envoyant f debut fin sur le vecteur [|f debut; f (debut + 1); ...; f fin|].

A.3 Définir une fonction conversion : string -> int vect envoyant une chaîne de caractère $m = x_0 \dots x_{n-1}$ sur le vecteur d'entiers correspondant [|numero_lettre x_0; ...; numero_lettre x_(n-1)|].

Exemple :

```
# conversion m;;
- : int vect = [|0; 1; 17; 0; 2; 0; 3; 0; 1; 17; 0|]
```

A.4 Nous utiliserons dans la dernière partie la structure de données pile, que l'on demande de réimplémenter, en partant de

```
# type pile = { mutable elements : int list };;
Type pile defined.
```

Reprogrammer les fonctions du cours cree_vide : unit -> pile, est_vide : pile -> bool, depile : pile -> int et empile : int -> pile -> unit (on rappelle que depile agit par effet de bord sur son argument).

Partie B – Première approche

On propose ici une première réponse au problème posé, de complexité peu satisfaisante. On comptera la complexité **dans le pire des cas**, en termes de nombre lectures et écritures dans un vecteur (make_vect sera considéré de coût nul), et d'opérations sur les chaînes de caractère, en nombre d'accès à un caractère, en supposant string_length de coût nul, sub_string m d p de coût p

B.1 Écrire une fonction facteurs : string -> int -> int vect vect telle que facteurs s a soit le vecteur des $n - a + 1$ facteurs de s, convertis en vecteurs d'entiers, donnés dans l'ordre croissant des indices.

Par exemple, on aura

```
# facteurs m 2;;
- : int vect vect =
 [|[|0; 1|]; [|1; 17|]; [|17; 0|]; [|0; 2|]; [|2; 0|]; [|0; 3|]; [|3; 0|];
   [|0; 1|]; [|1; 17|]; [|17; 0|]|]
```

B.2

a Écrire une fonction ClasseEq : string -> int -> int vect, n'utilisant pas de pile ni de liste, et comme unique vecteur auxiliaire facteurs m k, répondant à la question, c'est-à-dire envoyant m k sur le vecteur des indices des différentes classes d'équivalence pour E_k.

Dans notre exemple,

```
# ClasseEq m 1;;
- : int vect = [|0; 1; 2; 0; 3; 0; 4; 0; 1; 2; 0|]

# ClasseEq m 2;;
- : int vect = [|0; 1; 2; 3; 4; 5; 6; 0; 1; 2|]
```

b Donner la complexité de ClasseEq m k en fonction de k et de n (taille de m).

B.3 On propose ici un algorithme donnant les classes pour E_1, avec une complexité linéaire.

a Pour $i, j \in [\![0, n-1]\!]$, que signifie $< i, E_1, j >$?

b En utilisant un tableau auxiliaire de taille [2] 26, programmer une fonction construit_E1 ayant le même effet que function m -> ClasseEq m 1, mais avec une complexité linéaire.

Partie C – Une stratégie diviser pour régner

On souhaite améliorer notre réponse précédente,en adoptant une stratégie diviser pour régner pour k.

C.1

a Donner une expression construite sur E_a et E_b, équivalente à $< i, E_{a+b}, j >$.

b Donner, lorsque $b \leqslant a$, une expression construite sur E_a, équivalente à $< i, E_{a+b}, j >$.

C.2 On suppose désormais $b \leqslant a$, et on suppose provisoirement connues les p classes d'équivalence pour E_a, *i.e.* on suppose donné le vecteur v = ClasseEq m a, vecteur de taille $n - a + 1$ d'entiers entre 0 et $p - 1$.

Notre but est ici de construire le vecteur w de taille $n - (a + b) + 1$ donnant les classes d'équivalences pour E_{a+b}.

Nous allons pour ce faire utiliser des vecteurs de piles.

On suppose disposer d'une fonction cree_vect_pile_vide : int -> pile vect, qui envoie un entier naturel n sur le vecteur de taille n dont tous les termes sont la pile vide.

a Proposer, sans la programmer, une méthode pour construire un vecteur de piles dont les termes non vides sont constitués des classes d'équivalence pour la relation $\mathcal{R}$ donnée par $i\mathcal{R}j \Leftrightarrow < i + b, E_a, j + b >$. Malheureusement, pour construire aisément les classes d'équivalences pour E_{a+b} à partir de celles de E_a, il serait préférable que dans les classes pour la relation $\mathcal{R}$, représentées par des piles, et donc des listes d'entiers, les indices en relation pour E_a se succèdent (entre deux éléments i et j d'une même classe pour $\mathcal{R}$, et en relation pour E_a, ne se trouvent que des éléments en relation pour E_a avec i et j).

b Écrire une fonction empileClassesEa : int vect -> pile vect envoyant le vecteur v sur un vecteur de piles où les différentes piles non vides sont les différentes classes d'équivalence pour E_a.

c Écrire une fonction sousensembles : int vect -> pile vect -> int -> pile vect qui, lorsqu'elle prend, outre v et b, le vecteur de piles renvoyé à la question précédente en argument, envoie une pile de vecteurs où les différentes piles non vides sont les différentes classes d'équivalence pour E_{a+b}, tout en respectant le souhait formulé en C.2.a.

d En utilisant C.1.b, proposer une fonction classeAplusB : int vect -> pile vect -> int -> int vect qui, lorsqu'elle prend, outre v et b, le vecteur de piles renvoyé à la question précédente, fournit le vecteur cherché w.

Remarque : par souci de simplicité, on autorisera, à cette question et à la suivante, de numéroter les classes d'équivalence par leur plus petit.

e Écrire finalement la fonction classeAversAplusB : int vect -> int -> int vect qui calcule le vecteur w à partir du vecteur v et de b.

C.3 En utilisant une stratégie diviser pour régner, proposer une reprogrammation de ClasseEq.

2. *tip :* 26 est le nombre de lettres de notre alphabet ...