Mots de Lukasiewicz et recherche de motifs (Centrale 2008)

Durée : 4 heures

Ce contrôle est constitué de deux parties totalement indépendantes.

Partie I. Mots de Lukasiewicz

Dans cette partie, les mots considérés sont pris sur l'alphabet $\{-1, 1\}$.
On appelle *mots de Lukasiewicz* toute suite finie $u = (u_1, u_2, \ldots, u_n)$ à valeurs dans $\{-1, 1\}$ qui vérifie les deux propriétés suivantes :

$$\sum_{i=1}^{n} u_i = -1 \qquad \text{et} \qquad \sum_{i=1}^{k} u_i \geqslant 0 \text{ pour } 1 \leqslant k \leqslant n-1.$$

On notera avec un point $\cdot$ la concaténation de deux mots, par exemple, $a \cdot b$.
Par la suite, les mots de Lukasiewicz seront représentés en Caml par le type *int list*.
On demande d'indiquer le type de toutes les fonctions écrites.

I.1 Quelques propriétés

Question 1.1 Donner tous les mots de Lukasiewicz de longueur 1, 2 et 3, puis tous ceux de longueur paire.

Question 1.2 Écrire une fonction qui indique si un mot est de Lukasiewicz. Cette fonction renverra une valeur booléenne. La fonction proposée devra impérativement avoir une complexité en $O(n)$, où n est la longueur du mot d'entrée.

Question 1.3 Montrer que si u et v sont des mots de Lukasiewicz, alors $(+1) \cdot u \cdot v$ est un mot de Lukasiewicz.

Question 1.4 Réciproquement, montrer que tout mot de Lukasiewicz de longueur supérieure ou égale à 3 admet une décomposition unique de la forme $(+1) \cdot u \cdot v$, où u et v sont des mots de Lukasiewicz.

Question 1.5 Écrire une fonction `decompose` qui associe ce couple (u, v) à un mot de Lukasiewicz de longueur supérieure ou égale à 3. *On ne demande pas de traiter les cas où le mot fourni en entrée ne serait pas de Lukasiewicz.*

Question 1.6 On souhaite calculer tous les mots de Lukasiewicz d'une longueur donnée. Comparer les avantages d'une solution récursive appliquant le principe de la décomposition suggérée à la question 1.4, et celle d'une solution appliquant le même principe, mais pour laquelle on tabulerait les résultats intermédiaires.

Question 1.7 Écrire une fonction `obtenirLukasiewicz` qui calcule la liste des mots de Lukasiewicz de taille inférieure ou égale à un entier donné.

I.2 Dénombrement

Question 1.8 Soit $u = (u_1, \ldots, u_n)$ un mot tel que $\sum_{i=1}^{n} u_i = -1$. Démontrer qu'il existe un unique entier $i \in [\![1, n]\!]$ tel que $(u_i, u_{i+1}, \ldots, u_n, u_1, \ldots, u_{i-1})$ soit un mot de Lukasiewicz. Ce mot est appelé le *conjugué* de u.

Question 1.9 Écrire une fonction `conjugue` qui calcule le conjugué d'un mot $u = (u_1, \ldots, u_n)$ vérifiant $\sum_{i=1}^{n} u_i = -1$.

Question 1.10 En utilisant les résultats précédents, déterminer le nombre de mots de Lukasiewicz de longueur $2n+1$.
On pourra utiliser le résultat admis : *si u et v sont deux mots non vides, les deux propositions suivantes sont équivalentes :*

- $u \cdot v = v \cdot u$;
- *il existe un mot w et deux entiers $k, \ell \geqslant 1$ tels que $u = w^k$ et $v = w^\ell$.*

I.3 Capsules

On appelle *capsule* d'un mot u tout facteur de u de la forme $(+1,-1,-1)$.
On définit sur $\{-1,1\}^*$ une fonction ρ dite de *décapsulage* :

$$\rho(u) = u \quad \text{si } u \text{ ne contient pas de capsule ;}$$
$$\rho(u) = (u_1,\dots,u_{i-1},u_{i+2},\dots,u_n) \quad \text{si } (u_i,u_{i+1},u_{i+2}) = (+1,-1,-1) \text{ est la première capsule de } u.$$

Question 1.11 Justifier le fait que la suite $\left(\rho^n(u)\right)_{n\in\mathbb{N}}$ est constante au delà d'un certain rang. La valeur limite de cette suite sera notée $\rho^*(u)$.

Question 1.12 Écrire une fonction `rho` qui calcule $\rho(u)$.

Question 1.13 Ecrire une fonction `rholim` qui calcule $\rho^*(u)$.

Question 1.14 Démontrer enfin que u est un mot de Lukasiewicz si et seulement si $\rho^*(u) = (-1)$.

Partie II. Recherche de motif

Dans ce problème, nous allons étudier deux algorithmes de recherche de motif (en général noté p) dans un mot (en général noté m). Par exemple, le motif $p_0 = "bra"$ apparaît deux fois dans le mot $m_0 = "abracadabra"$. Les programmes de recherche de motif devront retourner la liste (éventuellement vide) des positions (au sens de Caml) du motif dans le mot. Dans l'exemple précédent, les programmes devront retourner une liste contenant les positions 1 et 8 (c'est la position de la première lettre qui est prise en compte). Plus formellement, si $m = m_0\cdots m_{n-1}$, on dit que p apparaît en position i dans m lorsque $p = m_i\cdots m_{i+|p|-1}$ avec $|p|$ la longueur de p.

II.1 Algorithme naïf

Question 2.1 Écrire une fonction `coincide` prenant en entrée deux chaînes de caractères `p` et `m`, une position `pos` et retournant `true` si `p` apparaît en position `pos` dans `m` (et `false` sinon). *Cette fonction devra uniquement utiliser des comparaisons de caractères, sans utiliser* `sub_string`. *De plus, en cas de réponse* `false`, *elle devra arrêter les tests dès que possible.*

Question 2.2 Écrire une fonction `recherche` prenant en entrée deux chaînes de caractères `p` et `m`, et retournant la liste (dans n'importe quel ordre, et éventuellement vide) des positions de `p` dans `m`.

Question 2.3 Évaluer la complexité (en termes de comparaisons de caractères, et en fonction de $|p|$ et $|m|$) de la fonction précédente dans le pire des cas. On exhibera un cas défavorable (en terme de complexité) avec p et m arbitrairement grands.

II.2 Algorithme de Rabin-Karp

La présentation de l'algorithme de Rabin-Karp est faite dans le cas de l'alphabet $A_0 = \{0,1,2,\dots,9\}$.
Un mot $m \in A_0^*$ peut être vu naturellement comme un entier ($m = 366$ est dans A_0^* mais aussi dans $\mathbb{N}$).

La première idée de cet algorithme est que, si on cherche $p = 366$ dans $m = 97463667305$, on va regarder les lettres de m par groupes de 3, en initialisant un compteur à $c = 974$ et en avançant dans m en ajoutant à chaque fois une nouvelle lettre et en effaçant la première de c. Dans notre exemple, c passe d'abord à 746 puis à 463. Plus formellement, lire la lettre $\ell = m_{i+|p|}$ dans m en effaçant m_i change c en : $10c + \ell - 10^{|p|}m_i$.
Si $c = p$, cela signifie que le motif p est présent en position i dans m.

On suppose dans cette première version de l'algorithme de Rabin-Karp que p est de très petite longueur, de sorte que le compteur c ne dépassera jamais la valeur du plus grand entier autorisé par Caml.
On considère définie une fonction appelée `numeral` prenant comme entrée un caractère de l'alphabet A_0 et retournant la valeur entière correspondante.

```
numeral : char -> int = <fun>
```

Par exemple, `numeral '5'` renvoie l'entier 5.

Question 2.4 Écrire une fonction prenant en entrée un mot m, une longueur ℓ, et retournant la valeur initiale du compteur, calculée en lisant les $\ell = |p|$ premières lettres de m. Dans l'exemple donné plus haut, sur l'entrée $(m, 3)$, la fonction doit retourner 974.

Question 2.5 Écrire enfin une fonction prenant en entrée un motif, un mot (supposé de taille supérieure au motif), et calculant la liste des positions dans m où le motif p est présent.

L'hypothèse quant à la longueur « faible » de p étant très restrictive, on modifie l'algorithme précédent en choisissant un entier q modulo lequel on calculera c. En lisant la lettre $\ell = m_{i+|p|}$ et en effaçant m_i, le nouveau compteur devient donc : $c' \leftarrow 10c' + \ell - 10^{|p|} m_i \pmod{q}$.
Lorsque $c = p$, on a $c' \equiv p \pmod{q}$. Mais réciproquement, lorsque $c' \equiv p \pmod{q}$, on est pas assuré d'avoir $c = p$. On regarde alors (avec l'algorithme naïf) si le facteur de m correspondant au compteur c calculé est égal à p. Si $c' \equiv p \pmod{q}$ mais $c \neq p$, on parle de « fausse-position ».

Question 2.6 Donner les valeurs successives de c' lors de la recherche de $p = 366$ dans $m = 97463667305$ avec $q = 9$.

Question 2.7 Écrire une fonction recherchant les positions d'un motif dans un mot, en appliquant l'algorithme de Rabin-Karp, avec un entier q donné en paramètre.

Question 2.8 Lors de la recherche de $p = 0001000$ dans $m = 000000000$ avec $q = 1000$, combien de cas de fausse-position va-t-on rencontrer ?

Question 2.9 Comparer la complexité dans le pire des cas de l'algorithme de Rabin-Karp et de l'algorithme naïf.
q est supposé tel que les calculs arithmétiques modulo q sont d'un coût constant. Le temps de calcul prendra donc en compte ces opérations arithmétiques et les comparaisons de caractères.

Question 2.10 En pratique, aura-t-on intérêt à prendre q plutôt petit ou plutôt grand ? Que peut-on alors espérer pour le temps de calcul de la recherche des occurrences de p dans m ?
On demande une justification informelle, le choix de l'entier q et l'évaluation de temps moyen de calcul étant deux choses très délicates…

♦ ♦
♦