90.04M

SESSION 2009

Filière MP (groupe I)

Épreuve commune aux ENS de Paris, Lyon et Cachan

INFORMATIQUE

Durée : 4 heures

L'usage de calculatrices électroniques de poche n'est pas autorisé.

Rotations et repliages de mots

Ce problème s'intéresse à diverses propriétés des mots soumis à des rotations et des repliages, ainsi qu'à certains algorithmes relatifs à ces transformations de mots.

Les deux parties sont indépendantes, ainsi que dans une certaine mesure les différentes sous-sections. Les questions non traitées peuvent être admises pour aborder les questions ultérieures.

La qualité de la rédaction ainsi que les justifications (corrections des algorithmes, complexités) seront des éléments d'appréciations. Il est également rappelé que les schémas ne sauraient à eux seuls tenir lieu de justification, mais qu'ils peuvent constituer une aide appréciable à la compréhension de la situation décrite et des notations employées.

Préliminaires

Conventions sur les mots.

On considère un alphabet A, c'est-à-dire un ensemble fini non vide, que l'on munit d'un ordre total $<$ sur les lettres de cet alphabet. Un certain nombre de questions utiliseront un alphabet particulier à deux éléments $A_2 = \{a, b\}$, muni de l'ordre suivant sur les lettres : $a < b$. Pour les questions mentionnant l'écriture binaire d'un nombre entier, l'alphabet $B = \{0, 1\}$ servira à noter les chiffres d'une telle écriture binaire.

Un mot formé sur l'alphabet A est une suite d'éléments de A indicée soit par $\mathbb{N}$ (pour un mot infini), soit par une partie de $\mathbb{N}$ de la forme $[\![0..n[\![$ (pour un mot fini). On note A^∞ (respectivement A^*) l'ensemble des mots infinis (respectivement finis) formés sur l'alphabet A. Pour tout mot u fini ou infini, $u[i]$ désignera la i-ème lettre de ce mot (en numérotant à partir de 0). **À partir de maintenant, tous les mots seront considérés finis sauf mention explicite du contraire.**

Le mot vide sera noté ε. On note $|u|$ la longueur d'un mot. Pour deux mots u et v, on note uv le mot obtenu par concaténation de u et v. Cette opération est associative et admet ε comme élément neutre (autrement dit, A^* muni de cette opération est un monoïde).

Un mot v est un préfixe d'un autre mot u s'il existe un mot w tel que $u = vw$. Un début de u est un préfixe de u distinct de u et de ε. Similairement, un mot v est un suffixe de u lorsqu'il existe un mot w tel que $u = wv$, et une fin de u est un suffixe distinct de u et de ε. On notera respectivement $Pre(u)$, $Deb(u)$, $Suf(u)$, $Fin(u)$ les ensembles de préfixes, de débuts, de suffixes et de fins du mot u.

Algorithmes et complexité.

Pour les questions demandant l'*écriture d'un algorithme en pseudo-code*, on utilisera un langage ou pseudo-langage au choix, avec les structures usuelles de données (tableaux, listes, etc.) et de contrôle (si, tant que, pour, etc.). Les calculs de complexité comptabiliseront le nombre d'opérations élémentaires effectuées dans le cas le pire : lecture ou écriture dans une variable ou une case de tableau, ajout ou suppression en tête de liste, opération arithmétique de base, etc. Sauf mention contraire, on ne cherchera pas à estimer les complexités exactement, mais seulement à en donner un ordre de grandeur en utilisant la notation asymptotique $O(...)$. Les indices d'un tableau T de taille n vont de 0 à $n-1$, et ses éléments sont notés $T[0], ..., T[n-1]$. Un tableau peut être de taille 0. On dispose d'une primitive LONGUEUR de complexité 1 retournant la taille d'un tableau, et d'une primitive CONCAT de complexité linéaire en ses arguments prenant deux tableaux en entrée et retournant un nouveau tableau formé par leur concaténation. La représentation des mots par des tableaux est encouragée, mais l'usage d'autres représentations est autorisé tant que cela ne pénalise pas les complexités obtenues.

Partie 1 : Rotation de mots et minimalité

Ordre lexicographique sur les mots. À partir de l'ordre total $<$ sur les lettres de l'alphabet, on obtient un ordre sur les mots comme suit :

- Pour deux mots u et v, on note $u \lhd v$ lorsqu'il existe un entier naturel i tel que $i < min(|u|, |v|)$ et $u[i] < v[i]$, ainsi que $u[j] = v[j]$ pour tout $j < i$.
- On note ensuite $u < v$ lorsque $u \in Pre(v) \setminus \{v\}$ ou $u \lhd v$.
- Enfin $u \leq v$ lorsque $u < v$ ou $u = v$.

On parlera d'*ordre lexicographique* (strict ou large) pour ces relations $<$ et $\leq$ sur les mots.

Question 1.1 *Montrer que la relation $\leq$ est une relation d'ordre totale sur les mots. Admet-elle de plus petit et de plus grand élément ? Écrire en pseudo-code un algorithme* EstPlusPetit *prenant en argument deux mots u et v et déterminant si $u < v$. Donner sa complexité.*

Rotation et minimalité. Un mot v est une rotation d'un autre mot u lorsqu'il existe deux autres mots non-vides w et r tels que $u = wr$ et $v = rw$. Un mot u sera dit *minimal* s'il est non-vide et que $u < v$ pour toute rotation v de u. On remarquera en particulier qu'un mot de longueur 1 est minimal.

1.1 Tests de minimalité

Question 1.2 *Écrire en pseudo-code un algorithme* EstMinimal1 *testant si un mot est minimal. Justifier sa correction et donner sa complexité.*

Question 1.3 *Soit u un mot tel que $Deb(u) \cap Fin(u) \neq \varnothing$. Montrer que u ne peut être minimal. Cette propriété suffit-elle à caractériser les mots non-minimaux ?*

Question 1.4 *Montrer qu'un mot non-vide u est minimal si et seulement si toute fin v de u est telle que $u < v$.*

Question 1.5 *En utilisant cette caractérisation, écrire en pseudo-code un algorithme* EstMinimal2 *qui teste si un mot u est minimal. Par rapport à* EstMinimal1*, déterminer le gain en nombre de comparaisons de lettres nécessaires dans le cas le pire.*

1.2 Énumération des mots minimaux

Question 1.6 *Montrer que l'ordre $\leq$ restreint aux mots minimaux admet un plus petit élément et un plus grand.*

Pour $n > 0$, on appelle n-minimal tout mot minimal de longueur inférieure ou égale à n, et on note $M(n)$ le nombre de mots n-minimaux.

Question 1.7 *Soient v et w deux mots minimaux tels que $v < w$. Montrer alors que vw est également minimal.*

Question 1.8 *Réciproquement, pour un mot u minimal tel que $|u| > 1$, prouver l'existence de deux mots minimaux v et w tels que $u = vw$ et $v < w$.*

Question 1.9 *À l'aide des questions précédentes, écrire en pseudo-code un algorithme* EnumMinimaux *prenant en entrée la liste des lettres de l'alphabet A dans l'ordre, ainsi que l'entier n, et retournant en sortie la liste de tous les mots n-minimaux dans l'ordre lexicographique. Justifier la correction de* EnumMinimaux *et donner sa complexité.*

Pour le reste de cette section, on utilise l'alphabet A_2.

Question 1.10 *Calculer alors $M(n)$ pour $0 < n \leq 5$. Montrer que tous les mots minimaux de longueur au moins 2 partagent la même première lettre, ainsi que la même dernière lettre. Pour $n > 0$, prouver l'encadrement suivant : $2 + n(n-1)/2 \leq M(n) \leq 2^{(n-1)} + 1$.*

Il existe en fait un algorithme direct permettant de passer d'une étape dans l'énumération des n-minimaux à l'étape suivante :

```
MinimalSuivant(entiers n et m, tableau T de taille n)
 1  ▷ Les m premières cases de T doivent former un mot n-minimal u ≠ b
 2  Pour i ← m à n − 1 Faire
 3      T[i] ← T[i modulo m]
 4  Fin Pour
 5  p ← n − 1
 6  Tant Que T[p] = b Faire
 7      p ← p − 1
 8  Fin Tant Que
 9  T[p] ← b
10  Retourner p + 1
11  ▷ Les p + 1 premières cases de T forment maintenant le mot
    n-minimal qui suit u
```

Question 1.11 *Montrer que toute utilisation de* MinimalSuivant *conforme aux conditions de la ligne 1 termine en retournant un entier q tel que $1 \leq q \leq n$. Déterminer la complexité de* MinimalSuivant.

Question 1.12 *Pour $n > 0$ et u un mot n-minimal différent de b, montrer que* MinimalSuivant *permet bien d'obtenir un mot n-minimal v, puis montrer que v est le mot n-minimal le plus petit (au sens de $<$) tel que $u < v$.*

1.3 Factorisation en mots minimaux

Soit u un mot non-vide. Une *factorisation minimale* de u est une suite finie de mots minimaux $u_0, \ldots, u_{p-1}$ tels que $u = u_0 \ldots u_{p-1}$ et $u_{i+1} \leq u_i$ pour tout $i < p-1$.

Question 1.13 *Écrire en pseudo-code un algorithme* Factorise *permettant d'obtenir une factorisation minimale de u. On pourra partir d'une décomposition de u en mots minimaux et faire progressivement évoluer cette décomposition vers une factorisation minimale. Déterminer la complexité de* Factorise

Question 1.14 *Démontrer que tout mot non-vide possède une unique factorisation minimale.*

Partie 2 : Repliage de mots

Pour une lettre α de l'alphabet A_2, on appelle renversement $\overline{\alpha}$ de α la lettre telle que $\overline{a} = b$ et $\overline{b} = a$. On étend cette opération de renversement à tout mot u sur l'alphabet A_2 de la manière suivante :

- $|\overline{u}| = |u|$.
- $\overline{u}[i] = \overline{u[j]}$ pour toute paire d'entiers naturels i et j tels que $i + j = |u| - 1$.

On s'intéresse désormais à une suite particulière de mots sur l'alphabet A_2, que l'on appellera *suite des repliages*. Cette suite $(u_i)_{i\in\mathbb{N}}$ est définie de la manière suivante :

$$\begin{cases} u_0 &= \varepsilon \\ u_{n+1} &= u_n a \overline{u_n} \quad \text{pour tout } n \geq 0 \end{cases}$$

Question 2.1 *Écrire en pseudo-code une procédure* NiemeRepliage *prenant en entrée un entier n et retournant le mot u_n. Donner la complexité de cette procédure.*

2.1 Une caractérisation alternative

Question 2.2 *Montrer que pour tout entier n, le mot u_n est exactement constitué des lettres d'indices impaires du mot u_{n+1} (prises de gauche à droite). Montrer également que les lettres d'indices pairs de u_{n+1} sont une alternance de a et de b. En déduire une procédure* NiemeRepliageBis *basée sur ces constatations et produisant le même résultat que* NiemeRepliage.

2.2 Repliages infinis

Chaque u_n étant un début du mot suivant u_{n+1}, il existe alors un mot infini u_∞ dont tous les u_n sont des préfixes : on peut par exemple le définir via la relation $u_\infty[i] = u_{i+1}[i]$ pour tout entier i.

Question 2.3 *À l'aide de la question précédente, écrire en pseudo-code un algorithme* RepliageInfini *calculant directement $u_\infty[i]$ en fonction de i sans passer par le calcul complet des u_n. Quelle est la complexité de* RepliageInfini *?*

Pour les trois prochaines questions, on considère l'alphabet $B = \{0, 1\}$. On identifie tout entier naturel i avec le mot sur l'alphabet B correspondant à l'écriture binaire de i. Pour la question qui vient, les bits de poids faibles sont placés à gauche. Par exemple, le nombre 6 correspond au mot 011, tandis que le nombre 0 correspond au mot vide.

Question 2.4 *Dessiner un automate fini déterministe (de transitions étiquetées par 0 et 1) acceptant précisément les mots correspondant aux entiers i tels que $u_\infty[i] = a$. Justifier brièvement la correction de cet automate.*

On suppose désormais que la représentation des nombres se fait avec les bits de poids faibles à droite (6 correspond maintenant à 110).

Question 2.5 *Dessiner et justifier brièvement l'automate fini déterministe correspond à cette nouvelle situation.*

Pour la question suivante, on fixe une certaine longueur k, et l'on considère des entiers *k-bornés* : n est k-borné si et seulement si $0 \leq n < 2^k$. Une fois choisi un ordre sur les bits, un nombre k-borné peut être identifié avec un mot sur B de longueur k. Par exemple, pour $k = 8$ et les bits de poids faibles à droite, le nombre 6 correspond au mot 00000110. On suppose disposer de plus des opérations primitives suivantes sur ces entiers k-bornés :

- *incr* : ajout de 1 modulo 2^k à un nombre k-borné
- *double* : doublement modulo 2^k d'un nombre k-borné
- *non* : inversion de tous les bits d'un nombre k-borné
- *et* : calcul du "et" bit-à-bit de deux nombres k-bornés
- *estnul* : vérification de la nullité d'un nombre k-borné

On notera que ces définitions sont indépendantes du choix de l'ordre des bits.

Question 2.6 *En composant les seules primitives précédentes, comment peut-on obtenir une fonction qui pour tout nombre k-borné i détermine si $u_\infty[i] = a$ ou non ? On pourra tout d'abord chercher à construire le nombre ayant un bit 1 à l'emplacement du 0 de plus faible poids dans i (s'il existe), et des bits 0 partout ailleurs.*

2.3 Repliages et géométrie

Soit φ la fonction qui à la lettre a associe le nombre complexe i et à la lettre b associe $-i$. À partir de u_∞, on définit une suite de points $(z_n)_{n\in\mathbb{N}}$ du plan complexe comme suit :

$$\begin{cases} z_0 &= 0 \\ z_1 &= 1 \\ z_{n+2} &= z_{n+1} + \varphi(u_\infty[n]) * (z_{n+1} - z_n) \quad \text{pour tout } n \geq 0 \end{cases}$$

On s'intéresse désormais aux propriétés de la courbe C obtenue en reliant les points successifs de cette suite.

Question 2.7 *On suppose donnée une primitive* TraceLigne *prenant les coordonnées cartésiennes (x_1, y_1) et (x_2, y_2) des extrémités d'un segment à afficher. Écrire en pseudo-code un algorithme* TraceCourbe *qui pour un entier n trace la courbe C jusqu'au point z_n.*

Question 2.8 *Soit p une puissance de deux. Déterminer quelle transformation géométrique permet d'obtenir la portion de C comprise entre z_p et z_{2p} à partir de celle comprise entre z_0 et z_p.*

Question 2.9 *Montrer qu'il existe une similitude qui pour tout entier n envoie z_{2n} sur z_n. En déduire en particulier que pour tout entier k, si $p = 2^k$ alors $z_p = (1+i)^k$.*

On appelle *point de contact* de la courbe C tout point z pour lequel il existe deux indices $n \neq m$ tels que $z = z_n = z_m$. De tels points de contact existent, par exemple $z_7 = z_{11} = i - 2$.

Question 2.10 *Soit un point de contact $z = z_n = z_m$ avec $n \neq m$. Montrer que n et m sont de même parité. En déduire que si n est non nul, alors $z_{n-1} \neq z_{m+1}$.*

Question 2.11 *Soit un point de contact $z = z_n = z_m$ avec $n \neq m$. Montrer que n et m sont non nuls et que les segments $z_{n-1}z_{m-1}$ et $z_{n+1}z_{m+1}$ sont les diagonales d'un carré de centre z.*

Question 2.12 *En déduire qu'un segment de la courbe C ne peut y apparaître qu'une unique fois, et que C ne passe qu'au plus deux fois par un point donné.*

* *
*

Note historique. Les mots minimaux de la partie 1 sont également appelés *mots de Lyndon.* L'algorithme MINIMALSUIVANT a été conçu par Duval en 1988. En 1992, Berstel et Pocchiola ont établi que le coût moyen de MINIMALSUIVANT est constant : visiter successivement tous les mots n-minimaux via $M(n)$ utilisations de cet algorithme a un coût proportionnel à $M(n)$ (on ne compte pas ici d'actions de sauvegarde ou d'affichage des mots n-minimaux rencontrés lors de ces itérations). Duval a par ailleurs proposé également un algorithme permettant d'effectuer la factorisation minimale d'un mot en temps linéaire, ce qui permet également de résoudre la question du test de minimalité en temps linéaire.

La courbe C de la partie 2 est nommée *courbe du Dragon* ou *courbe de Harter-Heighway.* Un article de Martin Gardner dans le Scientific American l'a fait plus largement connaître en 1967. Cette courbe fractale possède de nombreuses propriétés d'autosimilarité et de pavage du plan. À noter qu'il est possible de fabriquer son début simplement : il suffit de replier en deux une bande de papier plusieurs fois, toujours dans le même sens, puis d'ouvrir chaque pli à angle droit.