SESSION 2010

2nd concours

INFORMATIQUE

École normale supérieure de Lyon

Durée : 3 heures

Ce livret comprend 4 pages numérotées de 1 à 4

L'usage de calculatrices électroniques de poche à alimentation autonome, non imprimantes et sans document d'accompagnement, est autorisé. Cependant, une seule calculatrice à la fois est admise sur la table ou le poste de travail, et aucun échange n'est autorisé entre les candidats.

Recherche de chaînes de caractères

Le sujet comporte trois parties. Il est recommandé de lire l'intégralité du sujet avant de commencer la rédaction, et de traiter les questions dans l'ordre de l'énoncé. On pourra cependant aborder une question en admettant les résultats des questions précédentes. Les algorithmes pourront être écrits dans un langage au choix du candidat, en utilisant les structures de contrôle habituelles. Toutes les réponses devront être justifiées.

1 Préliminaires

On s'intéresse aux algorithmes résolvant le problème suivant :

- Étant données deux chaînes de caractères M et T, trouver la première occurrence de M dans T si elle existe.

Notations. On considère des chaînes sur un alphabet fixé noté Σ, de cardinal fini $|\Sigma| > 0$. Les chaînes sont représentées par des tableaux C, de taille $|C| \geq 0$, indexés à partir de 1. On supposera que la *chaîne vide* [ ] est l'unique chaîne de longueur 0.

Définition. Étant données deux chaînes non vides M et T, une *occurrence de M dans T* est un entier $i \in \{1, \ldots, |T| - |M| + 1\}$ tel que $T[i + k - 1] = M[k]$ pour tout $k \in \{1, \ldots, |M|\}$.

Ainsi, par « trouver la première occurrence de M dans T », on entend « trouver, s'il existe, l'indice dans T du premier caractère de la première occurrence de M dans T ». On étudie donc des fonctions $\textsc{Match}(M, T)$ réalisant la spécification suivante :

$$\textsc{Match}(M,T) = \begin{cases} i & \text{si } i \text{ est le plus petit entier tel que} \\ & T[i+k-1] = M[k] \text{ pour tout } k \in \{1,\ldots,|M|\} \\ 0 & \text{si } M \text{ n'a pas d'occurrence dans } T \end{cases}$$

Question 1.1. *Quelle doit être la valeur de* $\textsc{Match}(\texttt{abc}, \texttt{abdababc})$ *?*

Coût des algorithmes. On ne s'intéresse qu'aux coûts d'exécution en temps. De plus, on ramènera le plus souvent le coût d'une fonction $\textsc{Match}(M, T)$ au nombre d'accès à T.

Question 1.2. *Discuter cette dernière hypothèse : dans quelle(s) circonstance(s) est-elle pertinente, dans quelle(s) circonstance(s) ne l'est-elle pas ?*

2 Un algorithme naïf

Un algorithme « naïf » pour la recherche de chaînes de caractères est présenté à la figure 1. Dans toute la suite, on admettra que $\textsc{MatchNaïf}(M, T) = j > 0$ si et seulement si j est la première occurrence de M dans T.

Question 2.1. *Considérons l'exécution de* MatchNaïf *sur les chaînes*

T `abababacaba`
M `abaca`

Donner les valeurs des variables i et k à chaque sortie de la boucle ***tant que*** *(lignes 6 à 10).*

```
MatchNaïf(M, T)
1    lm := |M| ;
2    lt := |T| ;
3    pour i de 1 à lt − lm + 1
4    faire
5        k := 1 ;
6        tant que M[k] = T[k + i − 1]
7        faire
8            si k = lm alors retourner i ;
9            k := k + 1 ;
10       finfaire
11   finfaire
12   retourner 0 ;
```

Figure 1: Algorithme naïf

Question 2.2. *Montrer que si M et T sont non vides alors* MatchNaïf(M, T) *ne fait pas d'erreur de bornes (autrement dit, tous les accès à M et T sont valides).*

Question 2.3. *Donner le nombre d'accès à T effectués par* MatchNaïf *sur l'entrée*

T `aaaab`
M `ab`

Question 2.4. *Donner, en fonction de n, le nombre d'accès à T effectués par* MatchNaïf *sur les chaînes $M =$* `ab` *et $T =$* `a`^{n}`b` *(c'est-à-dire n occurrences de* `a` *suivies d'une occurrence de* `b`*).*

Question 2.5. *Donner, en fonction de $|T|$ et de $|M|$, une borne sur le nombre d'accès à T de* MatchNaïf(M, T) *qui est atteinte dans le pire cas.*

3 Algorithmes linéaires en $|T|$

Nous allons maintenant voir comment écrire des algorithmes recherche linéaires en $|T|$. L'idée est d'utiliser, au cours de la recherche, la connaissance de T issue des tests précédemment réussis. Cette idée est exploitée de différentes manières dans les algorithmes que nous allons voir. Ils ont cependant en commun de chacun reposer sur une phase de pré-traitement de T ou de M.

Notations. Si $1 \leq i \leq j \leq |C|$, on note $C[i \dots j]$ la chaîne $C[i] \dots C[j]$. Par exemple, si $C =$ `abdababc`, alors $C[3 \dots 6] =$ `daba`. On supposera que $C[i \dots j]$ est la chaîne vide si $j < i$ (avec éventuellement $j \leq 0$).

On dit que $C[1 \dots j]$ est un *préfixe* de C et que $C[i \dots |C|]$ est un *suffixe* de C. De plus, $C[i \dots |C|]$ est un suffixe *propre* de C si $i > 1$ (la chaîne vide est donc un préfixe et un suffixe propre de toute chaîne non vide). Par exemple, `abd` est un préfixe de `abdababc` et `abc` est un suffixe propre de `abdababc`.

3.1 Recherche par plus long suffixe

Dans un premier temps, on suppose qu'une phase de pré-traitement calcule, à partir de chaînes T et M données, un tableau S, de longueur $|T|$, tel que $S[i]$ est la longueur du plus long suffixe

de $T[1 \dots i]$ qui est un préfixe de M.

Question 3.1. *Montrer que $S[i] \leq |M|$ pour tout $i \in \{1, \dots, |T|\}$.*

Question 3.2. *Montrer que $S[i] = |M|$ si et seulement si $i - |M| + 1$ est une occurrence de M dans T.*

Question 3.3. *En supposant que l'on dispose de S, écrire un algorithme de recherche de chaînes de caractères linéaire en $|T|$ (on ne comptera que les accès aux tableaux).*

Notation. Si C est une chaîne et $x \in \Sigma$ est un caractère, on désigne par $C.x$ la chaîne de longueur $|C| + 1$ obtenue en ajoutant x à la fin de C.

On veut maintenant calculer S en utilisant une fonction $\textsc{Trans}(M, i, x)$, où $0 \leq i \leq |M|$ et $x \in \Sigma$, telle que $\textsc{Trans}(M, i, x)$ est la longueur du plus grand suffixe de $M[1 \dots i].x$ qui est un préfixe de M.

Question 3.4. *Donner les valeurs de* Trans *pour la chaîne $M =$ `abaca` et l'alphabet $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}\}$.*

Question 3.5. *Soit $i \in \{1, \dots, |T| - 1\}$.*

(i) Montrer que $S[i + 1] \leq S[i] + 1$.

(ii) Montrer que $M[1 \dots S[i + 1]]$ est un suffixe de $M[1 \dots S[i]].T[i + 1]$.

Question 3.6. *Donner une relation de récurrence pour S en fonction de* Trans *et de T.*

Jusqu'à la question 3.8 comprise, on supposera que pour une chaîne M et un alphabet Σ donnés, on obtient les valeurs de Trans en temps unitaire.

Question 3.7. *Donner un autre algorithme qui calcule S en fonction de* Trans *et T, et dont le nombre d'accès à T est linéaire en $|T|$.*

En mettant bout à bout les algorithmes des questions 3.7 et 3.3, on obtient un algorithme de recherche utilisant Trans et dont le coût d'exécution est linéaire en $|T|$.

Question 3.8. *Donner un algorithme de recherche linéaire en $|T|$, qui utilise* Trans *et qui évite le calcul explicite de S.*

Question 3.9. *Proposer une représentation mémoire de* Trans *et écrire un algorithme qui calcule* Trans *en fonction de Σ et de M. Donner le nombre d'accès à M en fonction de $|M|$ et de $|\Sigma|$. Commenter.*

3.2 Algorithme de Knuth-Morris-Pratt

Dans cette partie, on se propose d'étudier le célèbre algorithme de Knuth-Morris-Pratt (désigné par KMP dans la suite). C'est un algorithme linéaire en $|T|$, qui s'inspire des algorithmes de la partie 3.1, mais dont le pré-traitement est moins coûteux. L'idée est que l'on n'a pas besoin de calculer complètement Trans.

Question 3.10. *Soient deux chaînes T et M. Supposons que $i < |T|$ n'est pas une occurrence de M dans T.*

(i) Montrer qu'il existe un unique $k \in \{0, \dots, |M|\}$ tel que $M[1 \dots k] = T[i \dots i + k - 1]$ et $M[k + 1] \neq T[i + k]$.

```
KMP(M, T, SP)
1    m := |M| ;
2    t := |T| ;
3    i := 1 ;
4    k := 0 ;
5    tant que i + k ≤ t et k < m
6    faire
7        si T[i + k] = M[k + 1] alors k := k + 1 ;
8        sinon
9            si k = 0 alors i = i + 1 ;
10           sinon
11               i := i + k − SP[k] ;
12               k := SP[k] ;
13   finfaire
14   si k = m alors retourner i ;
15   sinon retourner 0 ;
```

Figure 2: Algorithme de Knuth-Morris-Pratt

(ii) Montrer que toute occurrence de M dans T supérieure à i est supérieure ou égale à $i+k-n$, où n est la longueur du plus grand suffixe propre de $T[i \dots i+k-1]$ qui est un préfixe de M.

On suppose que l'on dispose d'un tableau SP de longueur $|M|$ tel que $SP[k]$ est la longueur du plus long préfixe de M qui est un suffixe propre de $M[1 \dots k]$.

Question 3.11. *Calculer SP pour la chaîne $M =$* `abaca`.

On admettra que SP peut être calculé en temps linéaire en $|M|$. Une implémentation de KMP est présentée à la figure 2.

Considérons, pour une exécution de $\mathrm{KMP}(M, T, SP)$, les n premiers passages dans la boucle **tant que**, avec $n > 0$. Pour $p \in \{0, \dots, n\}$, notons i_p et k_p les valeurs respectives des variables i et k au $(p+1)$-ème passage à la ligne 5. On a donc $i_0 = 1$ et $k_0 = 0$.

Question 3.12. *Montrer que les suites $(i_p + k_p)_{0 \le p \le n}$ et $(i_p)_{0 \le p \le n}$ sont croissantes.*

Question 3.13. *Soit $p \in \{0, \dots, n-1\}$.*

(i) À quelles conditions sur les tests des lignes 7 et 9 a-t-on $i_{p+1} + k_{p+1} > i_p + k_p$?

(ii) À quelles conditions sur les tests des lignes 7 et 9 a-t-on $i_{p+1} > i_p$?

Question 3.14. *Donner une majoration linéaire en $|T|$ du nombre d'accès à T lors d'une exécution de $\mathrm{KMP}(M, T, SP)$.*

Question 3.15. *Montrer que pour tout $p \in \{0, \dots, n\}$, on a $T[i_p \dots i_p + k_p - 1] = M[1 \dots k_p]$.*

Question 3.16. *Montrer que pour tout $p \in \{0, \dots, n\}$, toute occurrence de M dans T est supérieure ou égale à i_p.*

Question 3.17. *Montrer que $\mathrm{KMP}(M, T, SP) = j > 0$ si et seulement si j est la première occurrence de M dans T.*