SESSION 2003

Filière MP (groupes MPI/MI)

Épreuve commune aux ENS de Paris et Cachan

Filière MP (groupe I)

Épreuve commune aux ENS de Paris, Lyon et Cachan

Filière PC (groupe I)

Épreuve commune aux ENS de Paris et Lyon

INFORMATIQUE

Durée : 4 heures

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.

Préambule

Le sujet comporte 4 parties.

La partie 1 *introduit différents problèmes d'algorithmique sur les mots et étudie la complexité des algorithmes naïfs.*

La partie 2 *propose un algorithme optimal pour le problème de la recherche de motifs.*

La partie 3 *introduit une structure de données importante dans de nombreux problèmes d'algorithmique des mots,* l'arbre des suffixes, *et montre comment le calculer en temps linéaire en la taille de l'entrée.*

La partie 4 *est destinée à l'étude de quelques applications de l'arbre des suffixes.*

Les parties 1, 2, 3 sont indépendantes. La partie 4 dépend de la partie 3, mais peut être traitée indépendamment en admettant les résultats qui y sont établis.

Le soin apporté à la rédaction et à la présentation sera apprécié par les correcteurs.

A. Notations et primitives de manipulation des mots. *Dans tout le problème, $\mathcal{A}$ est un alphabet fini, de cardinal σ fixé. On supposera que la comparaison de deux éléments de $\mathcal{A}$ (caractères) s'effectue en temps constant. La i-ème lettre d'un mot s sera notée $s[i]$. Le sous-mot constitué des caractères d'indice $i \leq l \leq j$ sera noté $s[i..j]$. On conviendra que $s[i..j]$ est le mot vide si $j < i$. La longueur d'un mot s sera notée $|s|$. La concaténation de deux mots u et v sera notée $u.v$; il s'agit du mot w de longueur $|u| + |v|$ tel que $w[i] = u[i]$ si $i \leq |u|$, $w[i] = v[i - |u|]$ sinon.*

On supposera que toutes ces primitives de manipulation des mots peuvent être effectuées en temps constant.

Si les entrées d'un problème ont pour taille $O(n)$, on supposera que toutes les opérations arithmétiques (addition, soustraction, multiplication, comparaison) sur des entiers inférieurs à n se font en temps 1.

B. Avertissement. *Cette épreuve est une épreuve d'informatique. À ce titre, la résolution des questions d'algorithmique du sujet sera prise en compte de façon importante dans l'évaluation de la copie. On attire tout particulièrement l'attention des candidats sur le fait que dans l'écriture d'algorithmes sur les mots, il convient d'être attentifs aux dépassements d'indice ; c'est-à-dire que si $m_1 \ldots m_k$ est un mot de taille k, il n'est pas licite d'appeler m_j pour $j > k$ ou $j \leq 0$ au cours d'un algorithme.*

Sauf mention expresse du contraire, on ne demande pas de justifier formellement les algorithmes écrits.

Outre les structures de contrôle habituelles, les structures de données du programme et les primitives les manipulant, les candidats pourront utiliser dans l'écriture de leurs algorithmes les primitives définies au paragraphe **A**.

⋆ ⋆ ⋆

1. Quelques problèmes d'algorithmique des mots

1. Écrire un algorithme COMPARE_SOUS_CHAÎNE(s, t, i, j, k) qui prend en argument deux mots de $\mathcal{A}^*$ et trois entiers positifs ou nuls, et qui renvoie `faux` si $i = 0$, $j = 0$. $i - k > |s|$, $j + k > |t|$ ou $s[i..i + k] \neq t[j..j - k]$, `vrai` sinon. Quelle en est la complexité ?

Recherche de motifs. *Soit $c \in \mathcal{A}^*$ un mot fixé de longueur m (le* motif*), et $t \in \mathcal{A}^*$ de longueur n (le* texte*). Le problème de la* recherche du motif *c dans le texte t consiste à trouver l'ensemble $\mathcal{I}_t$ des indices $i \in [1, n-m]$ tels que $t[i..i+m] = c$. Dans les algorithmes, cet ensemble pourra être représenté comme une liste sans répétitions.*

2. Montrer que l'ensemble des mots t tels que $\operatorname{card} \mathcal{I}_t \geq 1$ est un langage rationnel. L'ensemble des mots t tels que $\operatorname{card} \mathcal{I}_t = 0$ est-il un langage rationnel ?

3. Écrire un algorithme prenant en entrée les deux mots c et t et résolvant le problème de la recherche de motifs en temps $O(mn)$.

Plus longue sous-chaîne commune. *Soient s et t deux mots de $\mathcal{A}^*$ de longueurs respectives n et m.*

Une sous-chaîne commune *est un mot u de $\mathcal{A}^*$ de longueur l tel qu'il existe i et j avec $s[i..i+l] = t[j..j+l] = u$.*

Une plus longue sous-chaîne commune *est une sous-chaîne commune de longueur maximale.*

4. Proposer un algorithme de complexité $O(nm^3)$ trouvant une plus longue sous-chaîne commune de deux mots s et t.

Répétitions maximales. *Soit s un mot de $\mathcal{A}^*$ de longueur n. Un* triplet de répétition *de s est un triplet (i_1, i_2, j) tel que $1 \leq i_1 < i_2$, $i_2 + j \leq n$ et $s[i_1..i_1+j] = s[i_2..i_2+j]$.*

Un triplet de répétition (i_1, i_2, j) est dit maximal *si ni $(i_1 - 1, i_2 - 1, j+1)$, ni $(i_1, i_2, j+1)$ ne sont des triplets de répétition. On dit alors que $s[i_1..i_1+j]$ est une répétition maximale.*

5. Proposer un algorithme calculant toutes les répétitions maximales d'un mot s. Quelle est sa complexité ? On pourra, cette fois, représenter l'ensemble des répétitions comme une liste pouvant contenir plusieurs fois le même élément. Comment détecter les répétitions intervenant plusieurs fois dans la liste ?

* * *

2. Algorithme avancé pour la recherche de motifs

Pour tout élément t de $\mathcal{A}^*$, on appelera préfixe de t un mot u de longueur au plus $|t|$ tel que $u = t[1..|u|]$. Soit s un mot de longueur n. Pour $i \in [2, n]$, on note $L(i)$ la longueur du plus long préfixe de $s[i..n]$ qui est aussi un préfixe de s. On pose

$$d_i := \max_{1<j\leq i; L(j)\neq 0} (j + L(j) - 1),$$

on note g_i le plus petit indice atteignant le maximum. Si $L(j) = 0$ pour tout $1 < j \leq i$, on pose $g_i = d_i = 0$.

Le mot $s[g_i..d_i]$ est donc le préfixe de s commençant à gauche de i et se terminant le plus à droite possible.

1. Calculer les $L(i), g_i, d_i$ pour $\mathcal{A} = \{a, b, c, d\}$, $s = aabcaabdaaabc$.

2. On suppose $k > d_{k-1}$. Soit i le plus petit entier tel que $s[k+i-1] \neq s[i]$. Exprimer $L(k), g_k, d_k$ en fonction de i, k, g_{k-1}, d_{k-1}. On pourra distinguer le cas $i = 1$ et le cas $i > 1$.

3. Soit $k \geq 3$ un entier. On suppose que $k \leq d_{k-1}$.

a. Montrer que $s[k..d_{k-1}] = s[k - g_{k-1} + 1..L(g_{k-1})]$.

b. En déduire que si $L(k - g_{k-1} + 1) < d_{k-1} - k + 1$, alors $L(k) = L(k - g_{k-1} + 1)$, $d_k = d_{k-1}$, $g_k = g_{k-1}$.

c. Dans le cas contraire, soit $j = \min(\{|s|\} \cup \{l \geq d_{k-1}; s[j] \neq s[j - d_{k-1}]\})$. Exprimer $L(k)$, g_k et d_k en fonction de j et de k.

4. Déduire de **2.** et **3.** un algorithme calculant les $L(i)$, g_i, d_i et effectuant $O(n)$ comparaisons de caractères.

5. Soit $\# \notin \mathcal{A}$. En considérant $s = P\#T$, montrer que l'on peut trouver un algorithme trouvant le motif P dans le texte T en temps $O(m+n)$.

* * *

3. Arbre des suffixes

On définit un type de données arbre étiqueté comme étant un triplet formé de deux entiers et d'une liste d'arbres étiquetés. Une feuille est un arbre étiqueté pour lequel la liste est vide ; la liste vide sera notée $[\,]$. L'ajout d'un élément x en tête d'une liste l sera noté $x :: l$.

On supposera données les fonctions :

$$\mathsf{debut}(i,j,l) = i,\ \mathsf{fin}(i,j,l) = j,\ \mathsf{fils}(i,j,l) = l.$$

Intuitivement, les deux entiers i, j associés à un noeud de l'arbre correspondent au mot $s[i..j]$. Ce mot sera appelé l'étiquette du nœud.

L'étiquette depuis la racine d'un nœud est l'étiquette du chemin qui mène de la racine à ce nœud, ce dernier non compris dans le chemin. En particulier, l'étiquette depuis la racine de la racine elle-même est ε.

On dira qu'il existe un chemin d'étiquette γ dans l'arbre s'il existe un nœud $n = (n_i, n_j, [A_1, \ldots, A_k])$ tel que l'étiquette u de n depuis la racine soit un préfixe de γ, et que γ soit un préfixe de $u.s[n_i..n_j]$. On dira que le chemin d'étiquette γ se termine en position z du nœud n si $\gamma = u.s[n_i..z]$. Dans le cas particulier $\gamma = \varepsilon$, on dira le chemin d'étiquette ε se termine en position 0 de la racine.

On suppose donnée une fonction trouve qui prend en argument un arbre étiqueté, un mot s de $\mathcal{A}^*$ et deux entiers i et j, et renvoie un triplet (n, f, z) constitué d'un nœud interne n, d'un booléen f et d'un entier z tels que

- $f =$ `faux` s'il existe un chemin issu de la racine d'étiquette $s[j..i]$; dans ce cas, ce chemin se termine en position z du nœud n.
- $f =$ `vrai` sinon ; dans ce cas, s'il existe un chemin issu de la racine d'étiquette $s[j..i-1]$, le chemin se termine en position z du nœud n. Sinon, $n = (1, 0, [\,])$ et $z = 0$.

Dans le cas où plusieurs chemins conviennent, on supposera que trouve détecte l'un quelconque d'entre eux. On établira à la question 3 que ce n'est jamais le cas dans ce problème.

La complexité de la fonction trouve sera supposée proportionnelle au nombre de nœuds se trouvant sur le chemin.

On suppose enfin que le fait de modifier un nœud interne n d'un arbre A en n' modifie également l'arbre A en remplaçant dans ce dernier n par n'.

On donne alors les deux fonctions suivantes :

```
ajoute(A, s, i, j) =
    (B, flag, z) ← trouve(A, s, i, j).
    Si flag alors
        si z = fin(B) alors
            fils(B) ← (i, |s|, [ ]) :: fils(B)
        sinon
            fils(B) ← [(z + 1, fin(B), fils(B)), (i, |s|, [ ])]
```

```
                fin(B) ← z.
            finsi
        finsi.

Arbre_suffixe(s) =
    A ← (1, 0, [(1, |s|, [])])
    Pour i de 2 à |s| faire
            Pour j de 1 à i faire
                ajoute(A, s, i, j)
            finpour.
    finpour.
    Renvoyer A.
```

1. Que vaut Arbre_suffixe(`mississipi`) ? On pourra donner les arbres sous forme de représentation arborescente usuelle.

2. Montrer que pour $i \geq 2, 1 \leq j \leq i$, avant l'appel à ajoute$(A, s, i, j)$ il existe dans A un chemin d'étiquette $s[j..i-1]$.

3. Montrer que les arbres successifs construits par les appels à ajoute vérifient :
si $(u, v, [A_1, \ldots, A_k])$ est un nœud interne de A, alors pour tout $1 \leq l < m \leq k$, $s[\text{debut}(A_l)] \neq s[\text{debut}(A_m)]$. En déduire que pour tout mot $t \in \mathcal{A}^*$, il existe au plus un nœud interne ν d'étiquette depuis la racine e tel que e est un préfixe de t et t est un préfixe de $e.s[\text{debut}(\nu)..\text{fin}(\nu)]$.

4. Montrer que la fonction Arbre_suffixe effectue $O(|s|^3)$ opérations.

Dans la fonction Arbre_suffixe, *on dira qu'on est* au stade l de l'étape m *si* $i = m$ *et* $j = l$. *Dans la fonction* ajoute, *on dira qu'on est dans le cas 2 si* $flag$ *vaut vrai ; qu'on est dans le cas 1 s'il est faux, que* fils$(B) = []$ *et* $z = i$ *; qu'on est dans le cas 3 sinon.*

5. Montrer que si l'on est dans le cas 1 au stade l de l'étape m, alors on sera dans le cas 1 au stade l de toutes les étapes ultérieures.

6. Montrer que si on est dans le cas 3 au stade l de l'étape m, alors on sera dans le cas 3 dans les stades ultérieurs de l'étape m.

7. Comment modifier Arbre_suffixe pour qu'il effectue seulement $O(|s|^2)$ itérations ?

L'objectif est maintenant de modifier la structure de données pour permettre à la fonction trouve *de s'exécuter en temps constant.*

8. On suppose qu'on est dans le cas 2 au stade j de l'étape $i+1$, résultant en la création d'un nœud interne d'étiquette depuis la racine $s[j..i]$. Montrer qu'au plus tard à la fin du stade $j+1$ il existe un nœud interne d'étiquette depuis la racine $s[j+1..i]$.

On modifie la définition d'un arbre, en ajoutant un champ supplémentaire contenant un arbre ; ce champ est utilisé pour stocker un lien suffixe *: le nœud* ν_1, *dont l'étiquette depuis la racine est* $s[j..i]$ *a un lien suffixe vers l'éventuel nœud* ν_2 *d'étiquette depuis la racine* $s[j+1..i]$.
Étant donné un nœud ν, *on note* Prof(ν) *le nombre de nœuds sur le chemin menant de la racine à* ν.

9. Montrer que s'il existe un lien suffixe allant d'un nœud ν_1 à un nœud ν_2, alors $\text{Prof}(\nu_1) - \text{Prof}(\nu_2) \leq 1$.

10. Expliquer comment on peut modifier Arbre_suffixe pour qu'il opère en $O(|s|)$ opérations. On pourra construire et utiliser les liens suffixes pour passer d'un chemin d'étiquette $s[i..j]$ à un chemin d'étiquette $s[i+1..j]$. On pourra en outre rajouter dans la structure d'arbre étiqueté un arbre permettant de remonter d'un nœud interne à son père.

11. Soit $\# \notin \mathcal{A}$. Montrer que l'arbre associé au mot $s\#$ a exactement $n+1$ feuilles correspondant aux n suffixes de s et à $\#$.

*Cet arbre est l'*arbre des suffixes *du mot* s.

* * *

4. Applications de l'arbre des suffixes

1. Comment utiliser l'arbre des suffixes pour obtenir la liste des suffixes d'un mot s triés par ordre lexicographique en temps linéaire ?

2. Soit T un mot de longueur n. Montrer que le calcul d'un arbre de suffixes de T permet ensuite de trouver les k occurrences d'un mot P dans T en temps $O(n+k)$. Comparer avec la méthode de la partie 2.

3. Montrer que si (i_1, i_2, t) est un triplet de répétition maximal alors $T[i_1..i_1+t-1]$ est l'étiquette depuis la racine de deux nœuds distincts de l'arbre des suffixes.

En déduire qu'il y a au plus $O(n)$ répétitions maximales dans T.

4. Montrer que l'arbre des suffixes permet de déterminer l'ensemble des répétitions maximales en temps linéaire.

5. Expliquer comment une structure de données généralisant l'arbre des suffixes permet de résoudre le problème de la plus longue sous-chaîne commune en temps linéaire.

* * * * *