Informatique II

Ce problème aborde des questions d'algorithmique sur les mots centrées autour du concept de sous-mot. Les questions peuvent être résolues en admettant les résultats des questions précédentes. L'usage de la calculatrice est interdit.

Quelques définitions et notations

Dans tout le problème on considère que $\Sigma = \{\mathtt{a}, \mathtt{b}, \ldots\}$ est un alphabet fini fixé contenant au moins deux lettres.

La *longueur* d'un mot $u \in \Sigma^*$ est notée $|u|$. Le mot vide est noté ε et on a $|\varepsilon| = 0$. On utilise la notation exponentielle « u^n » pour représenter la concaténation de n copies de u, avec évidemment $u^0 = \varepsilon$ pour tout u.

On rappelle que si $u = u_1 u_2$ (c.-à-d. que u est la concaténation des mots u_1 et u_2) alors u_1 est un *préfixe*, u_2 est un *suffixe*, et $u_2 u_1$ est un *conjugué* de u (ce qu'on note $u \sim u_2 u_1$).

On réutilisera les notations d'intervalles telles que « $[n, m]$ » et « $]n, m[$ » pour des intervalles d'*entiers*. Par exemple, « $[0, 3[$ » représente l'ensemble $\{0, 1, 2\}$, tandis que « $[1, 1[$ » et « $[3, 1]$ » représentent tous deux l'ensemble vide.

Pour deux entiers $n, m \in \mathbb{N}$, un *plongement* h de $[1, n]$ dans $[1, m]$ est une injection qui respecte l'ordre, c.-à-d. telle que $h(i) < h(i+1)$ pour tout $i \in [1, n-1]$. Si h plonge $[1, n]$ dans $[1, m]$ et si $v = y_1 \ldots y_m$ est un mot de m lettres, alors $v \circ h$ est le mot de n lettres $u = x_1 \ldots x_n$ donné par $x_i = y_{h(i)}$. On dit que u est *extrait de v par h*. On note $u \sqsubseteq v$ quand il existe un plongement h tel que $u = v \circ h$, et on dit que u est un *sous-mot* de v.

Conseils pour la rédaction des algorithmes

a. Les algorithmes peuvent être donnés dans la notation de votre choix du moment qu'elle est suffisamment précise. On pourra supposer que les types de données utilisés sont munis des primitives dont vous avez besoin sans que vous ayez à les redéfinir : on demande simplement que soient rappelés clairement leur spécification. Par exemple, on pourra écrire

```
for i = 1 to u.length - 1 do
  if u(i) == u(i+1) then ...
```

en expliquant que `u.length` est la longueur de `u` (c.-à-d. du mot contenu par la variable `u`), et que `u(i)` est la lettre à la position `i` dans `u`.

b. À chaque fois que vous proposez un algorithme, il est indispensable que vous justifiez précisément sa correction et que vous donniez sa complexité en temps (en choisissant les paramètres pertinents et en estimant que les opérations élémentaires prennent un temps $O(1)$).

c. Il est permis, pour répondre à une question, d'utiliser un algorithme proposé en répondant à une question précédente, par exemple sous la forme d'un sous-programme. Dans de tels cas, votre analyse de complexité décomposera le coût en faisant apparaître la contribution de chaque partie.

d. Vos algorithmes sur les mots ne sont pas censés tirer parti du fait que Σ est fixé. Par exemple, on considérera que la façon la plus efficace de trier une liste de n lettres de Σ prend nécessairement un temps $O(n \log n)$.

1 La relation de sous-mot

a. On considère la relation $\sqsubseteq$ entre les mots de Σ^*. Montrez que c'est une relation d'ordre (c.-à-d. que $\sqsubseteq$ est réflexive, transitive, et antisymétrique).

b. Montrez que $\sqsubseteq$ est une pré-congruence pour la structure de monoïde (c.-à-d. que $u \sqsubseteq v$ et $u' \sqsubseteq v'$ implique $uu' \sqsubseteq vv'$, et que $\varepsilon \sqsubseteq u$ pour tous mots u, u', v, v').

c. Est-ce que les conjugués des sous-mots d'un mot u coïncident avec les sous-mots de ses conjugués ? (Justifiez votre réponse).

2 Algorithmes pour les sous-mots

a. Donnez un algorithme disant si $u \sqsubseteq v$ (pour deux arguments $u, v \in \Sigma^*$).
NB : Pour cette question, on attend un algorithme répondant en temps polynomial.

b. En adaptant votre réponse à la question précédente, donnez un algorithme calculant *lplssm*(u, v), la longueur du plus long suffixe de u qui soit sous-mot de v (le résultat est donc un entier compris entre 0 et $|u|$).

Expliquez comment on pourrait répondre à la question **2.a** en utilisant *lplssm*. Comparer la complexité des deux solutions.

c. Le *span* d'un plongement h tel que $u = v \circ h$ est la valeur $1 + h(|u|) - h(1)$ si $u \neq \varepsilon$. Par convention, on pose *span*$(h) = 0$ si $u = \varepsilon$. On note *minspan*(u, v) pour $\min\{\textit{span}(h) \mid u = v \circ h\}$ (en convenant que *minspan*$(u, v) = -1$ si u n'est pas sous-mot de v).
Donnez un algorithme calculant *minspan*(u, v).

d. Donnez un algorithme disant, pour deux mots $u, v \in \Sigma^*$, si il existe un entier $n \in \mathbb{N}$ tel que $u \sqsubseteq v^n$.

3 Nombre de plongements

Pour $u, v \in \Sigma^*$, on note $\binom{v}{u}$ pour le nombre de plongements h tels que $u = v \circ h$.

a. Calculez $\binom{\texttt{abbab}}{\texttt{ab}}$.

b. Calculez la valeur de $\binom{u}{u}$, de $\binom{u}{\varepsilon}$, et de $\binom{\varepsilon}{u}$ (en fonction de u).

c. Montrez que, pour tous mots $u, v \in \Sigma^*$ et lettres $x, y \in \Sigma$ avec $x \neq y$, on a les égalités $\binom{yv}{xu} = \binom{v}{xu}$ et $\binom{xv}{xu} = \binom{v}{u} + \binom{v}{xu}$.

d. Donnez un algorithme calculant $\binom{v}{u}$ pour deux mots $u, v \in \Sigma^*$.

e. Proposez une expression simple pour calculer $\sum_{u \in \Sigma^*} \binom{v}{u}$ en fonction de v, et prouvez que votre proposition est correcte.

f. Montrez que l'égalité et l'inégalité suivantes sont valides pour tous les mots :

$$\binom{v_1 v_2}{u} = \sum_{\substack{u_1, u_2 \in \Sigma^* \\ \text{tq } u_1 u_2 = u}} \binom{v_1}{u_1}\binom{v_2}{u_2} \qquad \binom{v}{u_1 u_2} \leq \sum_{\substack{v_1, v_2 \in \Sigma^* \\ \text{tq } v_1 v_2 = v}} \binom{v_1}{u_1}\binom{v_2}{u_2}$$

Donnez le plus petit exemple prouvant que l'inégalité peut être stricte.

4 Sous-mots et conjugués

On note $u \sim \sqsubseteq v$ quand il existe un mot $u' \in \Sigma^*$ tel que $u \sim u'$ et $u' \sqsubseteq v$.

a. Est-ce que $\sim\sqsubseteq$ est une relation d'ordre ? (Justifiez votre réponse).

b. Donnez un algorithme pour décider si $u \sim\sqsubseteq v$.

c. Montrez que, pour tous $u, v \in \Sigma^*$ et $n \in \mathbb{N}$, on a

$$u \sim\sqsubseteq v^n \text{ si et seulement si } 0 \leq \mathit{minspan}(u, v^{n+1}) \leq n|v|.$$

d. Donnez un algorithme pour décider si $u \sim\sqsubseteq v^n$ (pour des arguments u, v et n). Comparez la complexité de votre algorithme avec la complexité qu'on obtiendrait en réutilisant votre réponse à la question **4.b**.

5 Sous-mots de langages réguliers

Pour un langage $L \subseteq \Sigma^*$, on note $\uparrow L$ sa *fermeture vers le haut*, et $\downarrow L$ sa *fermeture vers le bas*, définis par

$$u \in \uparrow L \overset{\text{def}}{\Leftrightarrow} \exists v \in L, v \sqsubseteq u,$$
$$u \in \downarrow L \overset{\text{def}}{\Leftrightarrow} \exists v \in L, u \sqsubseteq v.$$

Comme d'habitude, on parle de « langage fermé » pour dire qu'il est égal à sa fermeture.

a. Soit $L = \{\mathtt{a}^n\mathtt{b}^m \mid n + m > 1\}$. Que sont $\uparrow L$ et $\downarrow L$?

Les *expressions régulières* sont données par la grammaire abstraite suivante :

$$e \ ::= \ e + e' \mid e.e' \mid e^* \mid \varepsilon \mid \emptyset \mid \mathtt{a} \mid \mathtt{b} \mid \cdots$$

Une expression régulière e dénote un langage $L(e) \subseteq \Sigma^*$ dont on ne rappellera pas ici la définition (mais qui est ce qu'on appelle un *langage régulier*). On note $e \equiv e'$ quand $L(e) = L(e')$: par exemple, $\emptyset^* \equiv \varepsilon$.

b. Montrez que si L est régulier, alors $\uparrow L$ et $\downarrow L$ sont réguliers.

c. Montrez que pour tout langage L il existe un plus grand langage fermé vers le haut contenu dans L (qu'on notera $L^{\supseteq\uparrow}$), ainsi qu'un plus grand langage fermé vers le bas contenu dans L (qu'on notera $L^{\supseteq\downarrow}$).

d. Montrez que si L est régulier alors $L^{\supseteq\uparrow}$ et $L^{\supseteq\downarrow}$ le sont aussi.

6 Expressions régulières simples

Un *atome* est une expression régulière a de la forme $x + \varepsilon$ avec $x \in \Sigma$, ou bien de la forme $(x_1 + \cdots + x_m)^*$ pour des lettres $x_1, \ldots, x_m \in \Sigma$. Un *produit* p est une concaténation $a_1.\cdots.a_l$ d'atomes. Une *expression régulière simple* (une « ERS ») est une somme $p_1 + \cdots + p_k$ de produits. On admet les sommes vides (si $k = 0$, alors $p_1 + \cdots + p_k \equiv \emptyset$) et les produits vides (si $l = 0$, alors $a_1 \ldots a_l \equiv \varepsilon$).

a. Montrez que si e est une ERS, alors $L(e)$ est fermé vers le bas.

b. Montrez que si e est une expression régulière, alors $\downarrow L(e)$ est un langage *simple*, c.-à-d. un langage qui peut être décrit par une ERS.

c. Soient p_1, p_2 deux produits et a_1, a_2 deux atomes. Montrez que $L(e_1.p_1) \subseteq L(e_2.p_2)$ si et seulement si l'une des trois conditions suivantes est remplie :
1. $L(e_1) \not\subseteq L(e_2)$ et $L(e_1.p_1) \subseteq L(p_2)$, ou
2. $e_1 = e_2 = x + \varepsilon$ pour un $x \in \Sigma$, et $L(p_1) \subseteq L(p_2)$, ou
3. e_2 est de la forme $(x_1 + \cdots + x_m)^*$, $L(e_1) \subseteq L(e_2)$, et $L(p_1) \subseteq L(e_2.p_2)$.

d. En déduire qu'on peut tester en temps quadratique si $L(p) \subseteq L(p')$ pour deux produits p et p'. (On donnera l'idée de l'algorithme et de son analyse de complexité, sans écrire l'algorithme explicitement. On précisera les hypothèses faites sur la structure de données utilisée pour représenter des ERS).

e. Montrez que pour tous produits $p, p_1, \ldots, p_n$, on a $L(p) \subseteq L(p_1 + \cdots + p_n)$ si et seulement si il existe $i \in [i, n]$ tel que $L(p) \subseteq L(p_i)$.

f. En déduire qu'on peut tester en temps quadratique si $L(e) \subseteq L(e')$ pour deux ERS e et e'.