C36211

3è ANNÉE

INFORMATIQUE I

DURÉE : 5 HEURES

Aucun document personnel n'est autorisé

L'usage de toute calculatrice est interdit

Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.

Le sujet comporte 4 pages

Ce problème aborde des questions d'algorithmique sur des mots « compressés ». 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 $A = \{a, b, \ldots\}$ est un alphabet fini fixé contenant au moins deux lettres.

La *longueur* d'un mot $u \in A^*$ est notée $|u|$. Le mot vide est noté ε et on a $|\varepsilon| = 0$. On rappelle que si $u = u_1u_2u_3$ (c.-à-d. que u est la concaténation des mots u_1, u_2 et u_3) alors u_1 est un *préfixe*, u_2 est un *facteur*, et u_3 est un *suffixe*, de u.

Pour $0 \leq k \leq |u|$, le k^{e} préfixe de u est composé des k premières lettres de u, et son k^{e} suffixe est composé de ses $|u| - k$ dernières lettres, de sorte que u coïncide toujours avec la concaténation de son k^{e} préfixe et son k^{e} suffixe. On utilise la notation exponentielle « u^n » pour représenter la concaténation de n copies de u, avec évidemment $u^0 = \varepsilon$ pour tout u. Plus généralement, ces mêmes notations sont utilisées pour des suites finies d'éléments d'un ensemble autre que A.

Pour un mot $u \in A^*$, et deux entiers $1 \leq n \leq m \leq |u|$, on note $u[n, m]$ le facteur allant de la n^{e} à la m^{e} lettre de u. Si $v = u[n, m]$ alors on dit que v apparaît à la position n dans u.

Conseils pour la rédaction des algorithmes

a. Les algorithmes peuvent être écrits 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 à condition que soit rappelée clairement leur spécification. Par exemple, on pourra se contenter d'écrire

```
nb := 0
for i = 1 to u.length do
  if u(i) = a then nb := nb+1 fi
done
return nb
```

pour fournir un algorithme qui calcule le nombre de lettres a dans un mot u. Il faudra spécifier que `u.length` est la longueur de `u` (c.-à-d. du mot contenu dans la variable `u`), et que `u(i)` est la lettre à la position `i` dans `u`, mais il ne sera pas nécessaire de programmer ces primitives.

b. À chaque fois que vous proposez un algorithme, il est indispensable que vous justifiiez précisément sa correction et que vous donniez sa complexité asymptotique en temps (en choisissant les paramètres pertinents, en estimant que les opérations élémentaires prennent un temps $O(1)$, donc en raisonnant à un facteur multiplicatif près).

c. Il est permis de réutiliser (par exemple sous la forme de sous-programmes) les algorithmes proposés lors des questions précédentes. Dans de tels cas, votre analyse de complexité doit faire clairement apparaître comment les différentes parties contribuent au coût total.

1 Mots compressés et manipulations de base

Un *mot compressé* est une suite $W = w_1 \dots w_n$ où chaque w_i est soit une lettre a_i de A, soit une paire d'entiers (b_i, e_i) satisfaisant une condition de *bonne formation* qui sera précisée plus bas. On note $|W|$ la longueur n de W.

Un mot compressé W représente un mot $\widetilde{W} \in A^*$, son *expansion*. Pour $W = w_1 \dots w_n$, on définit $\widetilde{W}$ comme la concaténation $u_1 \dots u_n$ des n mots donnés par

$$u_i = \begin{cases} a_i & \text{si } w_i = a_i \text{ est une lettre,} \\ (u_1 u_2 \dots u_{i-1})[b_i, e_i] & \text{si } w_i \text{ est une paire } (b_i, e_i). \end{cases}$$

Ainsi (b_i, e_i) représente la reprise d'un facteur (de longueur $e_i - b_i + 1$) du début $u_1 \dots u_{i-1}$ de l'expansion. On suppose que W est *bien formé*, c'est-à-dire que chaque paire (b_i, e_i) de W vérifie $1 \leq b_i \leq e_i \leq |u_1 \dots u_{i-1}|$.

Dans toute la suite du problème, et quand W est un mot compressé, on notera $l_k(W)$ pour $|u_k|$ et $L_k(W)$ pour $\sum_{i=1}^{k} l_i(W)$. Ainsi $L_n(W) = |\widetilde{W}|$.

Pour un mot compressé $W = w_1 \dots w_n$, et deux indices $1 \leq i, j, \leq n$, on dit que, dans l'expansion de W, le facteur u_i dépend du facteur u_j, noté $i \succ j$, lorsque $w_i = (b_i, e_i)$ est une paire et que $L_j(W) \geq b_i$ et $L_{j-1}(W) < e_i$. Notons que $i \succ j$ implique $i > j$. Pour $i = 1, \dots, n$, on définit inductivement la *hauteur* $h(W, i)$ de l'élément w_i par

$$h(W, i) = \begin{cases} 0 & \text{si } w_i = a_i \text{ est une lettre,} \\ 1 + \max_{j \prec i} h(W, j) & \text{si } w_i = (b_i, e_i) \text{ est une paire.} \end{cases}$$

Enfin, la hauteur $h(W)$ de W est $\max_{1 \leq i \leq n} h(W, i)$.

On note $W \equiv V$ quand W et V sont deux mots compressés ayant la même expansion, c.-à-d. sont deux compressions d'un même mot.

Remarque importante : pour éviter toute confusion, on précise ici que l'objectif du problème n'est à aucun moment de trouver la meilleure façon de compresser un mot. Il s'agit plus simplement de manipuler des mots représentés sous forme compressée sans construire systématiquement leur expansion.

a. Calculez l'expansion $\widetilde{W}$ et la hauteur $h(W)$ de $W = a\, b\, (1,2)\, (2,3)\, c\, (5,7)\, (4,8)\, (11,15)$.

b. Quelle est, en fonction de $n = |W|$, la valeur maximale de $|\widetilde{W}|$? Justifiez.

c. Donnez un algorithme `wellformed`(W) qui dit si une suite finie de **lettres** et de paires d'entiers W est bien un mot compressé bien formé.

d. Donnez un algorithme `zconcat`(W, W') qui, étant donnés deux mots compressés W et W', construit un mot compressé V tel que $\widetilde{V} = \widetilde{W}\,\widetilde{W'}$. Majorez $|V|$ en fonction de $|W|$ et $|W'|$.

e. Donnez un algorithme `zprefix`(W, k) qui, étant donné un mot compressé W et un entier $0 \leq k \leq |\widetilde{W}|$, construit un mot compressé V tel que $\widetilde{V}$ est le k^{e} préfixe de $\widetilde{W}$. Majorez $|V|$ en fonction de $|W|$.

f. Donnez un algorithme `ztail`(W) qui, étant donné un mot compressé W, construit un mot compressé V tel que $\widetilde{V}$ est le 1er suffixe de $\widetilde{W}$. Analysez $|V|$ en fonction du nombre de paires et du nombre de lettres apparaissant dans W.

g. On cherche un algorithme `zsuffix`(W, k) qui, étant donnés un mot compressé W et un entier $0 \leq k \leq |\widetilde{W}|$, construit un mot compressé V tel que $\widetilde{V}$ est le k^{e} suffixe de $\widetilde{W}$. Si on exige un algorithme en temps polynomial, il s'agit d'une question difficile qui sera abordée à la partie 4.

Pour le moment, supposons qu'on implémente `zsuffix` comme une simple boucle qui invoque k fois votre fonction `ztail`. Quelle est la complexité en temps de cette solution ? Peut-on majorer la taille $|V|$ du résultat qu'elle produit par un polynôme en $n = |W|$ et k ? (Remarque : il s'agirait alors d'une solution *pseudo-polynomiale*. Une solution réellement polynomiale serait bornée par un polynôme en n et $\log(k)$).

2 Recherche de motif

On considère maintenant un problème de recherche de motif. Soit $m \in A^*$ un mot de r lettres (appelé le motif) et $W = w_1 \dots w_n$ un mot compressé : on voudrait déterminer si m apparaît comme un facteur dans $\widetilde{W}$ sans devoir construire $\widetilde{W}$. On demande donc systématiquement des algorithmes en temps polynomial en n et r.

a. Donnez un algorithme `occurs_at`(m, p, W) disant si m apparaît dans $\widetilde{W}$ à la position p, c.-à-d. si $m = \widetilde{W}[p, p + r - 1]$.

b. Montrez que si $m \neq \epsilon$ apparaît dans $\widetilde{W}$, alors il existe deux entiers i et s, $1 \leq i \leq |W|$ et $0 \leq s < |m|$, tels que m apparaît dans $\widetilde{W}$ à la position $L_i(W) - s$.

c. Donnez un algorithme `zgrep`(m, W) disant en temps polynomial si m apparaît quelque part dans $\widetilde{W}$.

3 Mots compressés parfaits et motifs réguliers

On considère un automate fini $\mathcal{A} = (Q, A, \delta, I, F)$ sur l'alphabet A, où $I, F \subseteq Q$ sont les ensembles d'états initiaux et finaux respectivement, et où $\delta \subseteq Q \times A \times Q$ est en ensemble de triplets définissant les règles de transitions. En général $\mathcal{A}$ n'est donc pas déterministe, ni complet, ni émondé, ... On note $L(\mathcal{A})$ le langage accepté (ou « reconnu », ou « engendré ») par $\mathcal{A}$ et on se pose le problème de décider efficacement si $\widetilde{W} \in L(\mathcal{A})$.

a. On dit qu'un mot compressé $W = w_1 \dots w_n$ est *parfait* si toutes les paires $w_i = (b_i, e_i)$ qu'il contient sont telles que $b_i = L_j(W) + 1$ et $e_i = L_k(W)$ pour des indices $j \leq k < i$ (qui dépendent de i). Ainsi, le facteur u_i est la concaténation $u_j u_{j+1} \dots u_k$ de facteurs de $\widetilde{W}$.

Donnez un algorithme `accept_parfait`$(\mathcal{A}, W)$ disant si l'automate $\mathcal{A}$ reconnaît l'expansion d'un mot compressé parfait W. Justifiez sa correction de façon détaillée. Précisez la structure de données utilisée pour les automates finis et donnez la complexité asymptotique de votre algorithme en fonction de $|W|$ et des paramètres pertinents (à préciser) pour $\mathcal{A}$. (Comme précisé plus haut, on considérera que la taille de l'alphabet A est fixée et n'est pas un paramètre.)

b. Soient un mot compressé parfait $W = w_1 \dots w_n$ de longueur n et deux entiers b et e tels que $1 \leq b \leq e \leq L_n(W)$: ainsi, $W' \stackrel{\text{def}}{=} W(b, e)$ est bien un mot compressé. Montrez qu'il existe une suite V de longueur $O(n)$ telle que $W'' \stackrel{\text{def}}{=} WV$ est un mot compressé parfait équivalent à W'.

c. *Décrivez*[1] un algorithme transformant en temps polynomial un mot compressé W quelconque en un mot compressé parfait W' équivalent. Majorez $|W'|$ en fonction de $|W|$.

d. Décrivez un algorithme en temps polynomial `accept`$(\mathcal{A}, W)$ disant si un automate $\mathcal{A}$ reconnaît l'expansion d'un mot compressé W.

e. Soient $u, v \in A^*$ deux mots. On dit que $u = a_1 \dots a_n$ est un *sous-mot* de $v = b_1 \dots b_m$, noté « $u \sqsubseteq v$ », quand il existe une suite $1 \leq i_1 < i_2 < \dots < i_n \leq m$ telle que $a_\ell = b_{i_\ell}$ pour tout $\ell = 1, \dots, n$. De façon équivalente, $u \sqsubseteq v$ si on peut obtenir u à partir de v par l'effacement de certaines lettres.

Décrivez un algorithme en temps polynomial `zsubword`(u, W) disant si $u \sqsubseteq \widetilde{W}$ pour un mot $u \in A^*$ et un mot compressé W.

4 Suffixes compressés

a. Décrivez un algorithme en temps polynomial `zsuffix`(W, k) qui, étant donnés un mot compressé W et un entier $0 \leq k \leq |\widetilde{W}|$, construit un mot compressé V tel que $\widetilde{V}$ est le k^{e} suffixe de $\widetilde{W}$.

b. Pour cet algorithme, majorez $|V|$ en fonction de $|W|$.

[1] À partir d'ici, on ne demande plus d'écrire explicitement les algorithmes, simplement d'expliquer leur principe.

FIN DE L'ÉPREUVE