Second Concours à l'ENS Paris-Saclay
Interrogation d'informatique

4 heures

Le sujet se compose de deux problèmes indépendants : le premier aborde les langages formels et la calculabilité, alors que le second traite d'algorithmique et de programmation. Il est bien sûr demandé de traiter les deux problèmes. **Il est demandé aux candidats de composer chaque problème sur un ensemble de copies différentes.**

Il est autorisé d'admettre le résultat d'une question pour traiter la suite d'un problème.

La rigueur du raisonnement scientifique, mais aussi la présentation pédagogique des résultats obtenus, est un point capital dans l'évaluation de l'épreuve.

Problème 1 - Concaténation de langages

Soit Σ un alphabet fini et non vide. La longueur d'un mot $w \in \Sigma^*$ est notée $|w|$. Un **automate fini** $\mathcal{A}$ sur l'alphabet Σ est la donnée d'un tuple $\mathcal{A} = \langle Q, \delta, I, T \rangle$ où Q est un ensemble fini d'états, $\delta \subseteq Q \times \Sigma \times Q$ est l'ensemble des transitions, $I \subseteq Q$ est l'ensemble des états initiaux et $T \subseteq Q$ est l'ensemble des états terminaux. La transition $(p, \sigma, q) \in \delta$ est notée par la flèche $p \xrightarrow{\sigma} q$. Un **calcul** c de $\mathcal{A}$ est une séquence finie de transitions qui forment un chemin dans le graphe de $\mathcal{A}$, $q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} q_2 \cdots \xrightarrow{\sigma_n} q_n$: on note un tel calcul $c = q_0 \xrightarrow{\sigma_1\sigma_2\cdots\sigma_n} q_n$. L'étiquette de c est le mot $\sigma_1\sigma_2\cdots\sigma_n$ de Σ^*. Le calcul est acceptant si $q_0 \in I$ et $q_n \in T$. Le **langage** de $\mathcal{A}$ est le sous-ensemble $\mathcal{L}(\mathcal{A})$ de Σ^* qui contient l'ensemble des étiquettes des calculs acceptants de $\mathcal{A}$. Un tel langage est dit **régulier**. L'automate $\mathcal{A}$ est dit **déterministe** si I possède un unique élément et pour tout $p \in Q$ et $\sigma \in \Sigma$, il existe au plus un état $q \in Q$ tel que $(p, \sigma, q) \in \delta$.

Partie I — Carré et racine carrée

1. Montrer que le langage L_0, contenant les mots de la forme $a^n b a^m b^p$, avec $n, p \geq 0$ et $m \geq 1$, est un langage régulier. On donnera l'automate déterministe minimal le reconnaissant.

La racine carrée d'un langage L est définie par $\sqrt{L} = \{u \in \Sigma^* \mid uu \in L\}$.

2. Que vaut $\sqrt{L_0}$ (avec L_0 le langage de la question précédente) ? Est-ce un langage régulier ?

3. Montrer que pour tout langage régulier L, le langage $\sqrt{L}$ est régulier.

4. On rappelle qu'on peut définir la concaténation des langages L_1 et L_2 par

$$L_1 \cdot L_2 = \{w \mid \text{il y a une factorisation } w = uv \text{ telle que } u \in L_1 \text{ et } v \in L_2\}$$

On note donc L^2 le langage $L \cdot L$. Il est bien connu que si L est régulier, alors L^2 est régulier. La réciproque est-elle vraie ou fausse : le fait que L^2 soit régulier implique-t-il que L soit régulier ?

Partie II — Concaténation duale

On appelle factorisation d'un mot $w \in \Sigma^*$ toute décomposition de w en deux mots $u, v \in \Sigma^*$ tels que $w = uv$. On peut définir une **concaténation duale** de deux langages L_1 et L_2 en modifiant la définition précédente de la concaténation en remplaçant quantification existentielle par quantification universelle et conjonction par disjonction :

$$L_1 \odot L_2 = \{w \mid \text{pour toute factorisation } w = uv, u \in L_1 \text{ ou } v \in L_2\}$$

5. Montrer que, pour tous langages L_1 et L_2, $L_1 \odot L_2 = \overline{\overline{L_1} \cdot \overline{L_2}}$, où l'on note $\overline{L}$ le complément du langage L.

6. En déduire que la classe des langages réguliers est close par concaténation duale.

7. Montrer que la classe des langages finis est close par concaténation duale.

8. Considérons le langage $L = \Big(L_a \odot (L_b \cup L_{\text{even}})\Big) \cap \Big(L_b \odot (L_a \cup L_{\text{even}})\Big)$ où

$$\begin{aligned} L_a &= \{uav \mid u, v \in \{a,b\}^*, |u| = |v|\} \\ L_b &= \{ubv \mid u, v \in \{a,b\}^*, |u| = |v|\} \\ L_{\text{even}} &= \{aa, ab, ba, bb\}^* \end{aligned}$$

Montrer que tout mot de L est de longueur paire.

9. Montrer que $L = \{uu \mid u \in \{a,b\}^*\}$.

Notons $\mathcal{C}$ la plus petite classe de langages sur Σ contenant les langages algébriques (ceux reconnus par une grammaire algébrique, *context-free grammars*) et qui est close par intersection. **On admet que le langage L de la question précédente n'appartient pas à la classe $\mathcal{C}$.**

10. Montrer que la classe des langages algébriques n'est pas close par concaténation duale.

11. Montrer que la classe des langages récursivement énumérables (ceux acceptés par une machine de Turing) est close par concaténation duale.

Problème 2 - Gestion d'un cache

Nous considérons un système de mémoire composé de deux types de mémoire : la ***mémoire principale*** et la ***mémoire cache***. Chaque type peut stocker des pages mémoire d'une taille fixe (toutes les pages ont la même taille). Pour accéder aux données de la mémoire principale, le système doit les placer dans le cache. Lorsque la page demandée est déjà dans le cache, elle est directement lue. Sinon, la page demandée est extraite de la mémoire principale, puis mise dans le cache, ce qui ralentira la vitesse d'extraction des données.

On note k le nombre de pages que peut stocker la mémoire cache et N le nombre de pages contenues dans la mémoire principale. Les pages seront numérotées de 1 à N.

Une séquence $\sigma = \langle \sigma[1], \sigma[2], \ldots, \sigma[m] \rangle$ de demandes de pages (avec $\sigma[i] \in \{1, 2, \ldots, N\}$) arrive au système de mémoire. Les demandes sont traitées dans l'ordre donné par σ. Un algorithme de chargement de pages doit satisfaire les deux contraintes suivantes :

- **Contrainte de présence :** à chaque étape i de l'algorithme, la page $\sigma[i]$ doit être dans le cache. Si elle ne l'est pas, elle doit être placée dans le cache.
- **Contrainte de capacité :** lors de la mise en cache d'une nouvelle page, si le cache contient déjà k pages, alors l'algorithme doit choisir quelle page supprimer en fonction d'une certaine **stratégie**.

L'objectif est de concevoir une stratégie $\mathcal{A}$ de chargement de pages dans le cache de manière à minimiser le nombre de suppressions de pages dans le cache : la fonction à minimiser que l'on notera $\text{coût}_{\mathcal{A}}(\sigma)$ est donc le **nombre de suppressions de pages** dans le cache effectué par la stratégie $\mathcal{A}$ pour gérer la séquence σ de demandes de pages.

Plusieurs stratégies classiques existent :

- **LRU (Least-Recently-Used) :** supprimer la page dont la dernière demande est la plus ancienne ;
- **LFD (Longest forward distance) :** supprimer la page dont la demande suivante est la plus éloignée dans la séquence (dans le futur).

Exemple : Considérons la séquence de demandes $\sigma = \langle 1, 2, 3, 1, 2, 4, 5, 2, 3, 1 \rangle$ illustrée ci-dessous. Le cache peut contenir au plus $k = 3$ pages. La stratégie LFD agit comme suit. Elle place les pages 1, 2, 3 dans le cache lors des étapes **1**, **2** et **3**. Aux étapes **4** et **5**, elle ne fait rien, puisque les pages $1 = \sigma[\mathbf{4}]$ et $2 = \sigma[\mathbf{5}]$ sont déjà dans le cache. A l'étape **6**, comme le cache est plein, elle doit supprimer une page pour traiter la demande de la page $4 = \sigma[\mathbf{6}]$. Avant de traiter la demande, le cache contient les pages 1, 2 et 3. Les prochaines étapes où les pages 2 et 3 sont demandées sont les étapes **8** et **9**. Comme la prochaine demande de la page 1 se fait à l'étape **10**, la page 1 est la page (dans le cache) dont la demande suivante est la plus éloignée dans la séquence (dans le futur). La stratégie LFD supprime donc la page 1 pour la remplacer par la page 4. À l'étape **7**, elle supprime la page 4 qui n'est plus du tout demandée dans la suite. Ensuite, elle supprime la page 2 à l'étape **10**, qui est la page d'indice minimal parmi celles (les trois en l'occurrence) qui ne sont plus demandées dans la suite. Le coût total de cette stratégie est de 3 puisqu'il y a eu 3 suppressions de pages.

étape	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
$\sigma =$	$\langle$ 1,	2,	3,	1,	2,	4,	5,	2,	3,	1 $\rangle$
	1	1	1	1	1	④	⑤	5	5	5
cache		2	2	2	2	2	2	2	2	①
			3	3	3	3	3	3	3	3

Remarquons que la stratégie LRU supprimerait la page 3 pour la remplacer par la page 4 à l'étape **6** car sa dernière demande est la plus ancienne (à l'étape **3**).

Partie I — Implémentation des stratégies

1. En supposant toujours $k = 3$, exécuter les stratégies LFD et LRU sur la séquence de pages $\sigma = \langle 4, 1, 2, 3, 1, 2, 4, 1, 2, 3\rangle$.

2. Si $k \geq N$, donner le coût $\text{coût}_{\mathcal{A}}(\sigma)$ d'une stratégie $\mathcal{A}$ appliquée à une séquence σ.

3. Soit T un tableau de longueur k, trié en fonction d'un ordre $\leq$ sur ses éléments.

a) Écrire un algorithme en pseudo-code qui trouve la position d'un élément donné dans T. Quelle est sa complexité ?

b) Écrire un algorithme en pseudo-code qui supprime un élément minimum du tableau T, tout en insérant un nouvel élément dans le tableau qui doit rester trié. Quelle est sa complexité ?

c) Un tableau trié est-elle la meilleure structure de données pour chercher, insérer, supprimer un élément parmi k éléments ? Quelle(s) autre(s) structures de données pouvez-vous recommander, le cas échéant ?

4. En supposant que la séquence σ est donnée sous forme de tableau, écrire un algorithme en pseudo-code appliquant la stratégie LRU, de complexité $\mathcal{O}(m \log k)$. On pourra utiliser des structures de données classiques et leurs opérations (sans les réécrire). Vous justifierez le calcul de complexité.

5. Écrire de même un algorithme en pseudo-code pour la stratégie LFD en utilisant la question 3 et un pré-traitement de la séquence σ. Il est attendu un algorithme de complexité $\mathcal{O}(N + m \log k)$, que vous justifierez.

Partie II — Performance de la stratégie LFD

On suppose $N > k$ jusqu'à la fin du sujet.

6. Montrer que toute stratégie doit au moins supprimer une page pour gérer une séquence σ contenant exactement $k + 1$ pages différentes demandées.

7. Supposons $N = k + 1$ dans cette question.

a) Soient $i < \ell$ deux étapes où la stratégie LFD supprime une page du cache. Montrer que $i + k \leq \ell$.

b) En déduire que $\text{coût}_{\text{LFD}}(\sigma) \leq \lfloor \frac{m}{k} \rfloor$.

c) Pour tout entier n, donner une séquence σ de $m = nk + 1$ demandes telle que $\text{coût}_{\text{LFD}}(\sigma) = n$. Exécuter la stratégie LFD sur cette séquence.

8. Soient $\mathcal{A}$ une stratégie et i un entier. Montrer qu'il existe une stratégie $\mathcal{B}$ telle que

1. pendant les $i - 1$ premières étapes, $\mathcal{B}$ se comporte comme la stratégie $\mathcal{A}$;
2. à l'étape i (si elle existe), la stratégie $\mathcal{B}$ utilise la stratégie LFD pour remplacer la page supprimée ;
3. $\text{coût}_{\mathcal{B}}(\sigma) \leq \text{coût}_{\mathcal{A}}(\sigma)$ pour toute séquence σ de demandes.

Notons $\mathrm{OPT}(\sigma)$ le nombre minimum de suppressions qu'une stratégie peut faire pour gérer une séquence σ.
9. Montrer que la stratégie LFD est optimale, c'est-à-dire que $\mathrm{coût}_{\mathrm{LFD}}(\sigma) = \mathrm{OPT}(\sigma)$, pour toute séquence σ.

Partie III — Stratégies au fil de l'eau

10. Exécuter la stratégie LRU sur la séquence de pages $\sigma = \langle 1, 2, 3, 3, 4, 3, 4, 3, 4\rangle$ en supposant que $k = 3$.

On appelle **stratégie au fil de l'eau** toute stratégie qui décide à l'étape i **sans tenir compte des étapes futures**.
11. Les stratégies LRU et LFD sont-elles des stratégies au fil de l'eau ?

L'évaluation des performances des stratégies au fil de l'eau se fait par rapport à une stratégie optimale qui connait la séquence des demandes en entier (par exemple la stratégie LFD). Une stratégie $\mathcal{A}$ est donc évaluée par la quantité

$$\mathrm{éval}_{\mathcal{A}} = \sup_{\sigma} \frac{\mathrm{coût}_{\mathcal{A}}(\sigma)}{\mathrm{OPT}(\sigma)}$$

où la borne supérieure est prise sur toutes les séquences σ de demandes possibles. Cette quantité est toujours supérieure ou égale à 1, est égale à 1 lorsque la stratégie est optimale : plus la quantité est proche de 1, meilleure est la stratégie.

12. Pour tout entier k, donner une séquence σ telle que $\mathrm{coût}_{\mathrm{LRU}}(\sigma) = k \cdot \mathrm{OPT}(\sigma)$. *Indication : on pourra chercher σ de longueur $m = 2k$.*

13. Soit $\mathcal{A}$ une stratégie au fil de l'eau et k un entier. En supposant que $N = k + 1$, montrer qu'il existe une séquence σ telle que $k \cdot \mathrm{OPT}(\sigma) \leq \mathrm{coût}_{\mathcal{A}}(\sigma)$. *Indication : on pourra chercher σ de longueur $m = 2k$ et utiliser la question 7a.*

14. Qu'en déduit-on sur les stratégies au fil de l'eau ?