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.

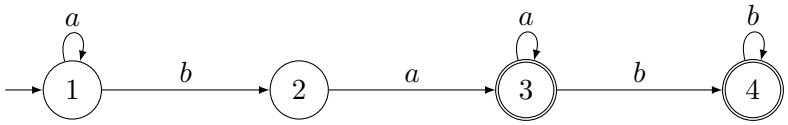
Concaténation de langages - Correction

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.

Solution : Ce langage est reconnu par l'automate déterministe suivant :



Cet automate est minimal puisque les langages reconnus par chaque état sont des résiduels distincts.

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 ?

Solution : Soit $v \in L_0$. Il existe $n, m, p \geq 0$ tels que $v = a^n b a a^m b^p$. Supposons que v soit un carré uu, avec $u \in \{a, b\}^+$ (puisque v est non vide). Si u commence par la lettre b, on a nécessairement $n = 0$. Le mot v doit alors contenir au moins deux b, donc $p > 0$: v est de la forme $baa^m b^p$. Puisque v contient au moins une lettre a, il doit en être de même pour u. Puisque v commence par ba, il doit donc aussi en être de même pour u. Mais cela est impossible, puisque les occurrences suivantes de b dans v ne sont jamais suivie d'une lettre a. Ainsi, u doit commencer par la lettre a. En particulier $n > 0$. Puisque v contient la lettre b, il doit en contenir deux. Donc $p > 0$. Le mot v, et donc aussi le mot u, doit donc finir par la lettre b. Puisque les lettres b en fin du mot v ne sont pas suivies de lettres a, on a nécessairement $u = a^n b$, d'où $m = n - 1$ et $p = 1$.
Réciproquement, les mots $u = a^n b$ avec $n > 0$ sont bien dans $\sqrt{L_0}$ puisque $u^2 = a^n b a^n b \in L_0$. Ainsi,

$$\sqrt{L_0} = a^+ b$$

qui est bien un langage régulier (puisque décrit par une expression régulière).

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

Solution : Soit $\mathcal{A} = \langle Q, \delta, I, T\rangle$ un automate reconnaissant le langage L. On construit un automate $\mathcal{A}' = \langle Q', \delta', I', T'\rangle$ reconnaissant $\sqrt{L}$: l'idée est de deviner un état $\widetilde{q}$ qui peut être atteint après avoir lu u dans $\mathcal{A}$ et tel qu'il existe aussi un calcul d'étiquette u menant de $\widetilde{q}$ à un état terminal. Pour cela, on pose $Q' = Q^3$ (le premier état permet le premier calcul sur le mot u, le second état est $\widetilde{q}$, le troisième état permet de simuler le second calcul $\widetilde{q}$ d'étiquette u), $I' = I \times \{(\widetilde{q}, \widetilde{q}) \mid \widetilde{q} \in Q\}$, $T' = \{(\widetilde{q}, \widetilde{q}, q) \mid \widetilde{q} \in Q, q \in T\}$ et

$$\delta' = \{\big((p, \widetilde{q}, p'), \sigma, (q, \widetilde{q}, q')\big) \mid \widetilde{q} \in Q,\ (p, \sigma, q) \in \delta, (p', \sigma, q') \in \delta\}$$

On peut montrer par récurrence sur la longueur de $u = \sigma_1\sigma_2\cdots\sigma_n$ que les calculs de $\mathcal{A}'$ sur u sont ceux de la forme

$$(q_0, \widetilde{q}, \widetilde{q}) \xrightarrow{\sigma_1} (q_1, \widetilde{q}, q'_1) \xrightarrow{\sigma_2} (q_2, \widetilde{q}, q'_2) \cdots \xrightarrow{\sigma_n} (q_n, \widetilde{q}, q'_n)$$

tels que $q_0 \in I$ et les calculs $q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} q_2 \cdots \xrightarrow{\sigma_n} q_n$ et $\widetilde{q} \xrightarrow{\sigma_1} q'_1 \xrightarrow{\sigma_2} q'_2 \cdots \xrightarrow{\sigma_n} q'_n$ sont valides dans $\mathcal{A}$. Un tel calcul est acceptant si et seulement si $q_n = \widetilde{q}$ et $q'_n \in T$, c'est-à-dire si et seulement la concaténation des calculs précédents

$$q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} q_2 \cdots \xrightarrow{\sigma_n} q_n \xrightarrow{\sigma_1} q'_1 \xrightarrow{\sigma_2} q'_2 \cdots \xrightarrow{\sigma_n} q'_n$$

sur le mot uu est acceptant dans $\mathcal{A}$. Cela prouve que $\mathcal{A}'$ reconnait le langage $\sqrt{L}$.

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 ?

Solution : La réciproque est fausse. Considérons le langage $L = \Sigma^* \setminus \{a^n b^n \mid n > 0\}$. Le langage L n'est pas régulier puisque son complémentaire ne l'est pas. Par ailleurs, puisque $\varepsilon \in L$, L^2 contient L. Mais puisque $\{a\}^*$ et $\{b\}^*$ sont inclus dans L, $\{a^n b^n \mid n > 0\} \subseteq L^2$. Ainsi, $L^2 = \Sigma^*$ est 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.

Solution : Montrons que pour tous langages L_1 et L_2, $L_1 \odot L_2 = \overline{\overline{L_1} \cdot \overline{L_2}}$. En effet, pour tout $w \in \Sigma^*$,

$$\begin{aligned} w \notin L_1 \odot L_2 &\iff \exists u, v \quad w = uv,\ u \notin L_1 \text{ et } v \notin L_2 \\ &\iff \exists u, v \quad w = uv,\ u \in \overline{L_1} \text{ et } v \in \overline{L_2} \\ &\iff w \in \overline{L_1} \cdot \overline{L_2} \\ &\iff w \notin \overline{\overline{L_1} \cdot \overline{L_2}} \end{aligned}$$

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

Solution : Puisque les langages réguliers sont clos par complémentaire et par concaténation, la famille des langages réguliers est close par concaténation duale.

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

Solution : Considérons deux langages finis L_1 et L_2. Soit m_i la longueur maximale d'un mot de L_i, pour $i \in \{1, 2\}$. Montrons que tous les mots de $L_1 \odot L_2$ ont une longueur majorée par $m_1 + m_2 + 1$. Supposons par l'absurde qu'il existe $w \in L_1 \odot L_2$ de longueur $|w| \geq m_1 + m_2 + 2$. Pour toute factorisation $w = uv$, $u \in L_1$ ou $v \in L_2$. Considérons la factorisation telle que $|u| = m_1 + 1$ et donc $|v| \geq m_2 + 1$. Alors, par définition de m_1 et m_2, $u \notin L_1$ et $v \notin L_2$, ce qui contredit l'hypothèse. Le langage $L_1 \odot L_2$ est donc fini.

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.

Solution : Tout d'abord, montrons que tout mot w de L est de longueur paire. Par l'absurde, supposons qu'un mot w de L s'écrive $u\sigma v$ avec $|u| = |v|$. Puisque $w \in L_a \odot (L_b \cup L_{\text{even}})$ et $\varepsilon \notin L_a$, la décomposition $\varepsilon \cdot u\sigma v$ implique que $u\sigma v \in L_b$, d'où $\sigma = b$. Symétriquement, $w \in L_b \odot (L_a \cup L_{\text{even}})$ implique $\sigma = a$, ce qui contredit l'hypothèse.

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

Solution : Considérons donc un mot $w = \sigma_1\sigma_2 \cdots \sigma_{2n} \in L$ de longueur paire. Puisque $w \in L_a \odot (L_b \cup L_{\text{even}})$, pour tout $1 \leq \ell \leq n$, la décomposition $w = uv$ avec $u = \sigma_1\sigma_2 \cdots \sigma_{2\ell-1}$ et $v = \sigma_{2\ell}\sigma_{2\ell+1} \cdots \sigma_{2n}$ implique que $u \in L_a$ ou $v \in L_b \cup L_{\text{even}}$. Le mot v est de longueur $2n - (2\ell) + 1 = 2(n - \ell) + 1$ impaire. Donc $u \in L_a$ ou $v \in L_b$, c'est-à-dire $\sigma_\ell = a$ ou $\sigma_{n+\ell} = b$. En particulier, si $\sigma_\ell = b$, alors $\sigma_{n+\ell} = b$. En considérant la même décomposition de $w \in L_b \odot (L_a \cup L_{\text{even}})$, on obtient que si $\sigma_\ell = a$, alors $\sigma_{n+\ell} = a$. On a donc $\sigma_\ell = \sigma_{n+\ell}$. Ceci étant vrai pour tout ℓ, w est de la forme uu. Donc $L \subseteq \{uu \mid u \in \{a, b\}^*\}$.
Finalement, montrons que tout mot de la forme uu appartient à L. On a déjà vu que les factorisations $u'v'$ de uu avec u' et v' de longueur impaire étaient valides vis-à-vis des deux concaténations duales dans L. Pour une factorisation $u'v'$ avec u' et v' de longueur paire (et donc $v' \in L_{\text{even}}$), c'est aussi bon du fait de la présence de L_{even} dans le membre droit de chaque concaténation duale dans L. Ainsi $L = \{uu \mid u \in \{a, b\}^*\}$.

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.

Solution : Puisque L n'est pas une intersection finie de langages algébriques, on en déduit que $L_a \odot (L_b \cup L_{\text{even}})$ ou $L_b \odot (L_a \cup L_{\text{even}})$ n'est pas un langage algébrique. Pourtant L_a et L_b sont des langages algébriques (la grammaire

$$S \to aSb \mid aSa \mid bSa \mid bSb \mid a$$

génère le langage L_a par exemple), et L_{even} est un langage régulier donc algébrique. La classe des langages algébriques étant close par union, elle ne peut donc pas être aussi 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.

Solution : Soient L_1 et L_2 des langages récursivement énumérables. Il existe donc deux machines de Turing M_1 et M_2 (à une seule bande) sur l'alphabet Σ, acceptant respectivement L_1 et L_2. On construit une machine M à trois bandes reconnaissant le langage $L_1 \odot L_2$. Pour un mot w sur son entrée, elle considère une par une toutes les factorisations $w = uv$ de w, en recopiant sur la seconde bande u et sur la troisième bande v (et en conservant la position séparant u et v dans la première bande, pour pouvoir itérer sur toutes les factorisations). Pour chaque factorisation, M simule M_1 sur la seconde bande et M_2 sur la troisième bande, jusqu'à ce que l'une des deux finisse par s'arrêter et accepte : dans ce cas, elle passe à la prochaine factorisation, ou elle accepte si elle était en train de vérifier la dernière factorisation. Si les deux machines s'arrêtent et rejettent, la machine M rejette à son tour.
Dans le cas où aucune des deux machines M_1 et M_2 ne s'arrête pour l'une des factorisations, ou si les deux rejettent, c'est que le mot w admet une factorisation $w = uv$ telle que $u \notin L_1$ et $v \notin L_2$: w n'est donc pas dans $L_1 \odot L_2$. Ainsi, la machine M reconnait bien le langage $L_1 \odot L_2$, qui est donc également récursivement énumérable.

Inspiré par l'article The dual of concatenation *de A. Okhotin, publié dans Theoretical Computer Science en 2005.*

Gestion d'un cache - Correction

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 $\rangle$
cache	1	1	1	1	1	④	⑤	5	5	5
		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 $\sigma = \langle 4, 1, 2, 3, 1, 2, 4, 1, 2, 3 \rangle$.

Solution : Pour la stratégie LFD, à l'étape 4, la page 3 doit être mise dans le cache. On supprime la page 4 puisque c'est celle dont la prochaine demande est la plus tardive.

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

Pour la stratégie LRU, à l'étape 4, on supprime la page 4 demandée il y a le plus longtemps :

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

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

Solution : Comme $k > N$, le cache peut contenir toutes les pages de la mémoire principale. Donc il n'y aura aucune suppression de pages. Donc le coût de toute stratégie est égale à 0.

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 ?

Solution : On suppose les tableaux indicés à partir de 1. On utilise une algorithme de recherche dichotomique.
Recherche_dichotomique(élément ℓ à rechercher, tableau trié de longueur k) :

(a) $m = \lceil k/2 \rceil$

(b) si $T[m] = \ell$, alors renvoyer m

(c) si $k = 1$ alors renvoyer « ℓ n'est pas dans le tableau »

(d) si $T[m] \geq \ell$, renvoyer Recherche_dichotomique$(\ell, T[m+1, \ldots, k])$

(e) si $T[m] \geq \ell$, renvoyer Recherche_dichotomique$(\ell, T[1, \ldots, m])$

La complexité de l'algorithme est en $\mathcal{O}(\log k)$ car à chaque appel, le nombre d'éléments dans la portion de tableau restant à traiter est divisé par 2.

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 ?

Solution :
Suppression_insertion(élément ℓ à insérer, tableau trié de longueur k) :

(a) $i = 2$

(b) tant que ($T[i] < \ell$ et $i \leq k$), faire $\{T[i-1] = T[i]\,;\, i = i+1\}$

(c) $T[i-1] = \ell$

(d) renvoyer T

La complexité de l'algorithme est en $\mathcal{O}(k)$. On peut faire des algorithmes plus compliqués (par exemple en recherchant l'élément ℓ avec une recherche dichotomique), mais la complexité dans le pire sera toujours linéaire du fait qu'il faut décaler une portion de tableau.

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 ?

Solution :
Non : il faut considérer par exemple un arbre binaire de recherche. Lorsqu'il est équilibré, les opérations de recherche, d'insertion et de suppression d'un élément peuvent toutes se faire en $\mathcal{O}(\log k)$ opérations, tout en maintenant l'équilibrage.

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.

Solution :
L'algorithme est le suivant :

À chaque étape i, faire

1. décider si $\sigma[i]$ est dans le cache
2. si oui, alors aller à l'étape suivante
3. sinon :
   (a) si le cache n'est pas plein alors insérer $\sigma[i]$ dans le cache
   (b) sinon
      i. déterminer la page e et son emplacement dans le cache
      ii. insérer la page $\sigma[i]$ dans le cache à la place de e

L'instruction *décider si* $\sigma[i]$ *est dans le cache* peut se faire en $\mathcal{O}(\log k)$ opérations : pour cela, il faut parcourir un arbre binaire de recherche équilibré qui maintient quelles sont les pages dans le cache.
Le test *le cache n'est pas plein* peut se faire en $\mathcal{O}(1)$ opérations : il suffit de stocker le nombre de pages que contient le cache à chaque instant.
L'instruction *déterminer la page* e *dans le cache* peut se faire en $\mathcal{O}(\log k)$ opérations : on associe à chaque page dans le cache son étape de dernière demande et on maintient le tout dans un second arbre binaire de recherche équilibré (ou une file de priorité stockée dans un tas binaire), de sorte qu'on peut rechercher la page associée à l'étape minimale dans le cache, la supprimer et insérer la nouvelle page à l'étape courante en $\mathcal{O}(\log k)$.
Au total la complexité est en $\mathcal{O}(m \log k)$ opérations.

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.

Solution : Pour l'initialisation :

1. Le pré-traitement correspond à la procédure suivante : remplir un tableau où $T[i]$ est la liste chainée de toutes les étapes où la page i est demandée, triée dans l'ordre croissant.

   Construire un tel tableau nécessite $\mathcal{O}(N)$ opérations : il suffit de parcourir la séquence de l'étape m à l'étape 1. Lors du traitement de l'étape i, il faut insérer au début de la liste correspondant à la liste du tableau $T[\sigma[i]]$ l'élement contenant i.
2. Il faut aussi initialiser un arbre binaire de recherche équilibré (ou une file de priorité stockée dans un tas binaire) tel que l'ordre des éléments $\ell < \ell'$ signifie que la valeur de tête de la liste de $T[\ell]$ est plus petite que celle de la tête de la liste de $T[\ell']$.

L'algorithme est le suivant :

A chaque étape i, faire

1. Supprimer la tête de la liste $T[\sigma[i]]$ *($\mathcal{O}(1)$ opérations)*
2. Décider si $\sigma[i]$ est dans le cache. *($\mathcal{O}(\log k)$ opérations)*
3. Si oui : supprimer la page $\sigma[i]$ dans l'arbre binaire de recherche *($\mathcal{O}(\log k)$ opérations)*
4. Sinon
   (a) si le cache est plein alors
      i. extraire l'élément e ayant la valeur maximun dans l'arbre binaire de recherche *($\mathcal{O}(\log k)$ opérations)*
      ii. insérer la page $\sigma[i]$ dans le cache à la place de e *($\mathcal{O}(1)$ opérations)*
   (b) sinon insérer la page $\sigma[i]$ dans une place vide. *($\mathcal{O}(1)$ opérations)*
5. insérer la page $\sigma[i]$ dans l'arbre binaire de recherche, associée à la valeur de la tête de la liste $T[\sigma[i]]$ *($\mathcal{O}(\log k)$ opérations)*

La complexité de l'algorithme est : $\mathcal{O}(N + m \log k)$.
On peut aussi diminuer le facteur N irréaliste en utilisant plutôt une table de hachage dans le pré-traitement.

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.

Solution : Soit t l'étape où l'on demande pour la première fois la $(k+1)$-ième page de σ. S'il n'y a pas déjà eu de suppression auparavant, le cache est plein. Donc il faut supprimer une page pour placer la page $\sigma[t]$.

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 $i + k \leq \ell$.

Solution : Comme $N = k + 1$, cela signifie qu'à l'étape i, toutes les autres pages sont dans le cache sauf la page $\sigma[i]$. Cela signifie que la prochaine fois qu'on supprime une page du cache, c'est lors d'une prochaine demande voulant accéder à la page $\sigma[i]$.
Par définition de la stratégie LFD, cela signifie qu'entre l'étape i et ℓ, il y a au moins une demande voulant accéder à toutes les autres pages que $\sigma[i]$. Si ce n'est pas le cas, la page $\sigma[i]$ n'est pas la page dont la demande suivante est la plus éloignée dans la séquence (dans le futur). Donc on a $i + k \leq \ell$.

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

Solution : Pour supprimer une page, il faut que le cache soit plein. Pour remplir le cache, il faut au moins k différentes demandes de pages. Donc la première suppression, si elle existe, apparait à une étape $i > k$. La précédente question montre que la stratégie LFD a besoin de supprimer une page au plus toutes les k étapes pour traiter σ. Ainsi, le pire des cas consiste à supprimer des pages pour insérer dans le cache $\sigma[k+1], \sigma[2k+1], \sigma[3k+1], \ldots$. Le nombre de suppressions est donc le plus grand entier j tel que $jk + 1 \leq m$, c'est-à-dire $j \leq (m-1)/k$. Puisque j doit être entier, $j \leq \lfloor (m-1)/k \rfloor \leq \lfloor m/k \rfloor$.

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.

Solution : Notons $p_1, \ldots, p_{k+1}$ les $k + 1$ pages de la mémoire principale.
Si n est pair, on considère

$$\sigma = \langle p_{k+1}, (p_1, p_2, \ldots, p_{k-1}, p_k, p_1, p_2, \ldots, p_{k-1}, p_{k+1})^{n/2} \rangle$$

A l'étape $k + 1$, le cache est plein et contient les pages $p_1, \ldots, p_{k-1}, p_{k+1}$. A cette étape il faut supprimer une page pour insérer p_k. Ce sera la page p_{k+1} puisque c'est celle dont la demande suivante est la plus éloignée dans la séquence. Donc la page p_{k+1} est supprimée pour mettre la page p_k. Le cache contient alors les pages $p_1, \ldots, p_k$. À l'étape $2k + 1$, il faut supprimer une page pour insérer p_{k+1}. C'est la page p_k qu'on supprime.
Plus généralement, une page est supprimée à toutes les étapes $ik+1$, pour $1 \leq i \leq n$:
— si i est pair, c'est la page p_{k+1} qui est supprimée pour mettre la page p_k ;
— si i est impair, c'est la page p_k qui est supprimée pour mettre la page p_{k+1}.
Au total, on a donc $n = \lfloor m/k \rfloor$ suppressions.
Si n est impair, on considère

$$\sigma = \langle p_{k+1}, (p_1, p_2, \ldots, p_{k-1}, p_k, p_1, p_2, \ldots, p_{k-1}, p_{k+1})^{\lfloor n/2 \rfloor}, p_1, p_2, \ldots, p_{k-1}, p_k \rangle$$

et on suit un raisonnement similaire.

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.

Solution : Soit σ une séquence de longueur supérieure à i.
A l'étape i, si $\mathcal{A}$ ne supprime aucune page, alors $\mathcal{B}$ se comporte comme $\mathcal{A}$ jusqu'à la fin de la séquence. Sinon, $\mathcal{B}$ supprime dans le cache la page q dont la demande suivante est la plus éloignée dans la séquence, comme la stratégie LFD l'y oblige. Soit p la page qui est supprimée par $\mathcal{A}$ pour mettre la page $\sigma[i]$ dans le cache à l'étape i.
Si $p = q$, alors $\mathcal{B}$ se comporte comme $\mathcal{A}$ jusqu'à la fin de la séquence. Par la suite, nous supposerons que $p \neq q$. Après l'étape i, $\mathcal{B}$ traite les demandes restantes comme $\mathcal{A}$ jusqu'à ce que la page p soit à nouveau demandée :

1. Si $\mathcal{A}$ supprime la page q du cache, alors $\mathcal{B}$ supprime la page p du cache.
2. Si $\mathcal{A}$ supprime la page x du cache avec $x \neq p$, alors $\mathcal{B}$ supprime cette même page x du cache.

Il est facile de remarquer que les caches de deux stratégies $\mathcal{A}$ et $\mathcal{B}$ contiennent un ensemble de pages de la forme $X' \cup \{q\}$ et $X' \cup \{p\}$ respectivement jusqu'à que $\mathcal{A}$ supprime la page q du cache.
Après l'étape où $\mathcal{A}$ supprime la page q du cache, les caches contiennent les mêmes pages pour les deux stratégies. Ensuite, $\mathcal{B}$ se comporte comme $\mathcal{A}$ pour les étapes restantes.
Remarquons que $\mathcal{B}$ se comporte comme $\mathcal{A}$ jusqu'à ce que la page p soit demandée. Considérons l'étape où la page p est demandée. Supposons que $\mathcal{A}$ n'a pas supprimé la page q du cache. À cette étape, le cache de $\mathcal{B}$ contient p tandis que celui de $\mathcal{A}$ ne contient pas p. La stratégie $\mathcal{A}$ doit supprimer une page dans le cache. Ce n'est pas le cas pour la stratégie $\mathcal{B}$.

- Si $\mathcal{A}$ supprime la page q du cache, alors le cache des deux stratégies contiennent les mêmes pages. Ensuite, $\mathcal{B}$ se comporte comme $\mathcal{A}$ pour les étapes restantes. La stratégie $\mathcal{A}$ aura fait une suppression de page de plus que la stratégie $\mathcal{B}$.
- Sinon lorsque la page q sera demandée, le cache de $\mathcal{B}$ ne contient pas la page q tandis que celui de $\mathcal{A}$ contient la page q. Donc, la stratégie $\mathcal{B}$ devra faire une suppression de page contrairement à la stratégie $\mathcal{A}$. À partir de cette étape, les caches contiennent les mêmes pages pour les deux stratégies. Ensuite, $\mathcal{B}$ se comporte comme $\mathcal{A}$ pour les étapes restantes. Les stratégies $\mathcal{A}$ et $\mathcal{B}$ auront fait le même nombre de suppressions de page.

Dans tous les cas, on voit que $\mathcal{B}$ ne fait pas plus de suppressions que $\mathcal{A}$ d'où $\text{coût}_{\mathcal{B}}(\sigma) \leq \text{coût}_{\mathcal{A}}(\sigma)$.

Notons $\text{OPT}(\sigma)$ 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 $\text{coût}_{\text{LFD}}(\sigma) = \text{OPT}(\sigma)$, pour toute séquence σ.

Solution : On procède par récurrence, en montrant pour tout $i \geq 0$ qu'il existe une stratégie $\mathcal{B}$ de coût $\text{coût}_{\mathcal{B}}(\sigma) \leq \text{OPT}(\sigma)$ telle que pendant les i premières étapes, $\mathcal{B}$ supprime les pages en fonction de la stratégie LFD.
Pour le cas de base $i = 0$, il suffit de considérer pour $\mathcal{B}$ une stratégie optimale pour la séquence σ. Pour l'hérédité, on applique la question précédente pour ajouter une étape supplémentaire d'utilisation de la stratégie LFD tout en faisant décroitre le coût de la stratégie.

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 $k = 3$.

Solution :

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

Solution : Oui pour la stratégie LRU. Non pour la stratégie LFD.

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

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

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 $\text{coût}_{\text{LRU}}(\sigma) = k \cdot \text{OPT}(\sigma)$. *Indication : on pourra chercher σ de longueur $m = 2k$.*

Solution :
Considérons $k + 1$ pages $p_1, \ldots, p_{k+1}$ de la mémoire principale. Soit

$$\sigma = \langle p_1, \ldots, p_k, p_{k+1}, p_1, p_2, \ldots, p_{k-1}\rangle$$

Pour la stratégie LRU,
- après l'étape k, il y a les pages $p_1, \ldots, p_k$ dans le cache ;
- à l'étape $k + 1$, la page p_1 est supprimée pour placer la page p_{k+1} ;
- à l'étape $k + 2$, la page p_2 est supprimée pour placer la page p_1 ;
- et ainsi de suite : à toutes les étapes suivantes, on doit supprimer la page qui est demandée à l'étape suivante.

Donc $\text{coût}_{\text{LRU}}(\sigma) = k$. Pour la stratégie LFD,
- après l'étape k, il y a les pages $p_1, \cdots, p_k$ dans le cache ;
- à l'étape $k + 1$, la page p_k est supprimée pour placer la page p_{k+1}, puisque c'est celle demandée le plus loin dans le futur (puisqu'elle n'est plus demandée) ;
- et après aucune n'est supprimée.

Donc $\text{OPT}(\sigma) = \text{coût}_{\text{LFD}}(\sigma) = 1$. En conclusion, $\text{coût}_{\text{LRU}}(\sigma) = k \cdot \text{OPT}(\sigma)$.

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 $k \cdot \text{OPT}(\sigma) \leq \text{coût}_{\mathcal{A}}(\sigma)$. *Indication : on pourra chercher σ de longueur $m = 2k$ et utiliser la question 7a.*

Solution : Construisons la séquence σ. On commence par demander les $k+1$ pages dans les $k+1$ premières étapes. Le stratégie $\mathcal{A}$ doit donc supprimer une page q_{k+1} à l'étape $k+1$. On demande cette page à l'étape $k+2$. À nouveau, $\mathcal{A}$ doit donc supprimer une nouvelle page q_{k+2} pour recharger q_{k+1}. On continue ainsi en demandant la page que la stratégie $\mathcal{A}$ vient de supprimer, pendant k étapes. Ainsi, $\text{coût}_{\mathcal{A}}(\sigma) = k$. La stratégie LFD exécute également une suppression à l'étape $k+1$. Mais la longueur de σ étant $m = 2k$, d'après la question 7a, la stratégie LFD ne supprimera plus aucune page jusqu'à la fin de la séquence. Ainsi, $\text{OPT}(\sigma) = \text{coût}_{\text{LFD}}(\sigma) = 1$. D'où $k \cdot \text{OPT}(\sigma) = \text{coût}_{\mathcal{A}}(\sigma)$.

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

Solution : Si $N = k+1$, alors l'évaluation de toute stratégie $\mathcal{A}$ au fil de l'eau vérifie $\text{éval}_{\mathcal{A}} \geq k$, de par la question précédente. Pour $k > 1$, aucune stratégie au fil de l'eau n'est donc optimale.