Exemple d'exercices - Epreuve d'Informatique, Filière MPI

Algorithmes de plus courts chemins

Question 1. On considère l'algorithme de Dijkstra basé sur une file de priorité. Donner et justifier sa complexité.

On définit un graphe $G = (V, E)$ avec pondération sur les arêtes $w \colon E \to \mathbb{R}$. On rappelle l'algorithme de Bellman-Ford, qui part d'une source s.

Algorithme 1 Algorithme de Bellman-Ford

```
1: procédure BellmanFord(G, w, s)
2:     d[s] ← 0                                  ▷ Distance depuis le sommet source s
3:     pour chaque sommet v ∈ V \ {s} faire
4:         d[v] ← +∞                             ▷ Initialisation des distances
5:     répéter |V| − 1 fois
6:         pour chaque arête u → v ∈ E faire
7:             si d[u] + w(u → v) < d[v] alors
8:                 d[v] ← d[u] + w(u → v)        ▷ Relaxation
9:     renvoyer d                                ▷ Renvoie les distances les plus courtes
```

Question 2. Montrer la correction de l'algorithme de Bellman-Ford ; autrement dit, si le graphe ne contient pas de cycle de poids négatif, l'algorithme détermine bien un plus court chemin pour chaque nœud à partir d'une source donnée. Donner la complexité de cet algorithme.

Question 3. Comment cet algorithme peut permettre de détecter les cycles de poids négatifs ?

On suppose à présent que G n'a pas de cycle de poids négatif. On connecte un nouveau nœud s_0 à tous les nœuds de V avec un poids de 0 et on exécute l'algorithme de Bellman-Ford à partir de s_0 sur le nouveau graphe orienté $G' = (V', E')$, où $V' = V \cup \{s_0\}$. Celui-ci détermine les distances $h(u) \triangleq d_{G'}(s_0, u)$ de s_0 à chaque nœud u de V', appelées *potentiels*. Ensuite on construit une nouvelle pondération w' sur les arêtes, définie par $w'(u \to v) = w(u \to v) + h(u) - h(v)$.

Question 4. Montrer que $h(u)$ est défini et négatif ou nul pour tout $u \in V'$.

Question 5. Montrer que les nouveaux poids w' sont tous positifs ou nuls.

Question 6. Montrer que les plus courts chemins pour G' avec pondération w' correspondent aux plus courts chemins pour G avec pondération w.

Question 7. En déduire une méthode pour calculer les plus courts chemins pour toutes les paires de nœuds dans G. Comparer sa complexité avec celle de l'algorithme de Floyd-Warshall.

Question 8. On considère à présent l'algorithme A*. Montrer que si l'heuristique choisie est monotone, on ne visite chaque sommet qu'une seule fois.

Éléments de correction

Question 1. Pour chaque arête on peut modifier une valeur dans la file de priorité, qui contient $|V|$ ou $|E|$ sommets. $O(|E|T_{\texttt{update}} + |V|T_{\texttt{extract}})$ où $T_{\texttt{update}}$ et $T_{\texttt{extract}}$ sont les complexités de mise à jour d'une priorité et extraction du minimum. À noter que $\log|E| \leqslant 2\log|V|$. Nous avons accepté $O((|E|+|V|)\log|V|) = O(|E|\log|V|)$ car le graphe peut être supposé connexe, ou éventuellement $O(|E| + |V|\log|V|)$ avec un tas de Fibonacci.

Question 2. Après k itérations on obtient le poids minimal d'un chemin de longueur au plus k (noté $d_k[u]$ par exemple). Si nul cycle n'est de poids négatif, le poids minimal est atteint par un chemin sans cycle, donc de longueur au plus $|V|-1$. La complexité est manifestement $O(|V||E|)$.

Question 3. On détecte un cycle de poids négatif en faisant une itération de boucle supplémentaire. Si une distance diminue, alors il y a un cycle de poids négatif, car il fournit un chemin de poids inférieur.

Question 4. C'est au plus nul car le candidat est l'arête de poids 0 de q à u. C'est défini car il n'y a pas de cycle de poids négatif.

Question 5. Supposons qu'il existe une arête $u \to v$ telle que $w'(u \to v) = w(u \to v) + h(u) - h(v) < 0$. Alors, on aurait trouvé un candidat $h(u) + w(u \to v) < h(v)$, ce qui contredit la définition de $h(v) \triangleq d_{G'}(s_0, v)$.

Question 6. Somme télescopique. Tous les chemins de a à b sont décalés d'une quantité constante $h(a) - h(b)$, donc les plus courts chemins sont conservés.

Question 7. On fait une exécution de l'algorithme de Bellman-Ford $O(|V||E|)$ puis $|V|$ exécutions de l'algorithme de Dijkstra sur les poids mis à jour pour toutes paires de nœuds. Complexité totale $O(|V||E|\log|V|)$, en général mieux que Floyd-Warshall $O(|V|^3)$. (Si le graphe est complet, on peut préférer l'implémentation de l'algorithme de Dijkstra original sans tas en $O(|V|^2)$.) C'est l'*algorithme de Johnson*.

Question 8. Si on note h' l'heuristique, alors elle est monotone si : $\forall u \to v \in E, h'(u) \leqslant w(u \to v) + h'(v)$. A* fait un parcours sur les priorités $dist(u) + h'(u)$ ce qui est implicitement un Dijkstra sur le graphe de coûts réduits $w(u \to v) + h'(v) - h'(u) \geqslant 0$ (dans l'autre sens, i.e. $h' = -h$). Dijkstra ne visite chaque sommet qu'une seule fois.

Mots réduits et réductions

Soit $\mathbb{A} = \{a_1, \overline{a_1}, a_2, \overline{a_2}, \ldots, a_n, \overline{a_n}\}$ un alphabet à $2n$ lettres. On dit qu'un mot w sur $\mathbb{A}$ est *réduit* s'il évite tout facteur de la forme $a_k\overline{a_k}$ ou $\overline{a_k}a_k$. On dit aussi que w est une *réduction* d'un mot x si x s'obtient à partir de w en ajoutant des facteurs de la forme $a_k\overline{a_k}$ ou $\overline{a_k}a_k$, éventuellement emboîtés les uns dans les autres ; si, en outre, w est réduit, on dit que c'est la *réduction complète* de x.

Question 1. Donner une réduction complète de chacun des mots suivants :

$$a_1\,\overline{a_2}\,a_3\,\overline{a_2}\,a_2\,\overline{a_2} \qquad a_1\,\overline{a_2}\,a_3\,\overline{a_1}\,a_2\,\overline{a_2}\,a_1 \qquad a_1\,\overline{a_2}\,a_3\,\overline{a_1}\,a_2\,\overline{a_3}\,a_1.$$

Question 2. Proposer un algorithme qui, étant donné un mot w quelconque, calcule une réduction complète de w en temps linéaire en $|w|$.

Question 3. Démontrer que tout mot admet une unique réduction complète.

On note désormais $\mathsf{red}(w)$ l'ensemble des réductions d'un mot w, et $\mathsf{red_c}(w)$ la réduction complète de w. Plus généralement, si L est un langage, on note $\mathsf{red}(L) = \bigcup_{w\in L} \mathsf{red}(w)$ l'ensemble des réductions des mots de L, $\mathsf{red_c}(L) = \{\mathsf{red_c}(w) \colon w \in L\}$ l'ensemble des réductions complètes des mots de L, et $\mathsf{red}^{-1}(L) = \{x \in \mathbb{A}^* \colon \mathsf{red}(x) \in L\}$ l'ensemble des mots qui se réduisent en un mot de L.

Question 4. L'ensemble $\mathsf{red_c}(\mathbb{A}^*)$ est-il rationnel ?

Question 5. Soit L un langage rationnel. L'ensemble $\mathsf{red}^{-1}(L)$ est-il nécessairement rationnel ?

Question 6. Soit L un langage non contextuel. L'ensemble $\mathsf{red}^{-1}(L)$ est-il nécessairement non contextuel ?

Question 7. Soit L un langage rationnel. Les ensembles $\mathsf{red}(L)$ et $\mathsf{red_c}(L)$ sont-ils nécessairement rationnels ?

Question 8. Soit L un langage non contextuel. Démontrer qu'il existe un entier $n \geqslant 1$ pour lequel tout mot $w \in L$ de longueur $|w| \geqslant n$ admet une factorisation $w = rstuv$ telle que $s \neq \varepsilon$ et $rs^ktu^kv \in L$ pour tout $k \geqslant 0$.

Question 9. Pour tout mot $w = w_1w_2\cdots w_k$, on note $\overline{w}$ le mot $\overline{w}_k\overline{w}_{k-1}\cdots\overline{w}_1$. Soit Σ l'alphabet $\{a_1, a_2, a_3, a_4\}$, puis φ le morphisme défini sur Σ par $\varphi(a_1) = a_3^4$, $\varphi(a_2) = a_1^4$, $\varphi(a_3) = a_2^4$ et $\varphi(a_4) = a_1a_2a_3a_4$. Démontrer que le langage

$$L = \{\varphi(w)\,\overline{w} \colon w \in \Sigma^*\}$$

est non contextuel, mais que $\mathsf{red}(L)$ et $\mathsf{red_c}(L)$ sont contextuels.

Éléments de correction et attentes de l'examinateur

Question 1. Il suffit de supprimer les motifs interdits tant qu'il en reste. En procédant ainsi, on obtient

$$a_1\,\overline{a_2}\,a_3\,\overline{a_2} \qquad a_1\,\overline{a_2}\,a_3 \qquad a_1\,\overline{a_2}\,a_3\,\overline{a_1}\,a_2\,\overline{a_3}\,a_1.$$

Cette question a pour but de s'assurer que le candidat a bien compris les notions de réduction et de réduction complète, ou alors de lui permettre de le faire.

Les candidats qui avaient mal compris l'énoncé, par exemple parce que la notion de facteurs emboîtés n'était pas claire, n'ont pas été pénalisés ici.

Question 2. Il suffit d'utiliser un algorithme glouton, qui consiste à parcourir w de gauche à droite et, pour chacun des préfixes ainsi rencontrés, d'en stocker la réduction complète. En particulier, pour tout mot $w \in \mathbb{A}^*$ et toutes lettres a et b dans $\mathbb{A}$, en utilisant les notations qui seront introduites plus bas (et en posant abusivement $\overline{\overline{a_k}} = a_k$) :

- si $\mathsf{red_c}(w) = \varepsilon$, alors $\mathsf{red_c}(wb) = b$;
- si $\mathsf{red_c}(w)$ est un mot non vide ua pour lequel $a = \overline{b}$, alors $\mathsf{red_c}(wb) = u$;
- si $\mathsf{red_c}(w)$ est un mot non vide ua pour lequel $a \neq \overline{b}$, alors $\mathsf{red_c}(wb) = uab$.

Si l'on note $w_{\leqslant i}$ le préfixe de w de longueur i, et si chaque mot est stocké sous forme d'une pile, on transforme donc $\mathsf{red_c}(w_{\leqslant i})$ en $\mathsf{red_c}(w_{\leqslant i+1})$ en temps constant, ce qui permet de calculer $\mathsf{red_c}(w)$ en temps linéaire en $|w|$.

Cette question a pour but d'évaluer les candidats sur une question d'algorithmique simple, puisque cette partie du programme ne reviendra pas dans la suite du sujet. Elle requiert d'être un minimum imaginatif, et d'être capable de proposer une procédure systématique là où, à la question précédente, le candidat a pu procéder de manière intuitive sans se demander comment il détectait les motifs à supprimer.

Question 3. Tout mot w admet une réduction de longueur minimale, qui sera nécessairement complète. On en démontre donc l'unicité par récurrence sur $|w|$.

Si w est réduit, il est sa seule réduction complète ; on suppose donc w non réduit. Soit x et y deux réductions complètes de w, puis $w = x_0 \to x_1 \to x_2 \to \cdots \to x_k = x$ et $w = y_0 \to y_1 \to y_2 \to \cdots \to y_\ell = y$ deux suites de mots dont chacun est obtenu à partir du précédent en supprimant un facteur interdit (c'est-à-dire de la forme $a\overline{a}$). Puisque x_k et y_ℓ sont les réductions complètes de x_1 et y_1, il s'agit de démontrer que x_1 et y_1 ont une réduction commune, disons z, dont x_k et y_ℓ seront également la réduction complète.

- Si x_1 et y_1 ont été obtenus en supprimant le même facteur, ils sont égaux, et on pose $z = x_1 = y_1$.
- S'ils ont été obtenus en supprimant deux facteurs ayant une lettre commune, c'est qu'ils ont été obtenus en remplaçant un facteur $a\overline{a}a$ de w par la seule lettre a : on a supprimé le facteur $a\overline{a}$ de gauche ou le facteur $\overline{a}a$ de droite. Dans les deux cas, on peut alors poser $z = x_1 = y_1$.
- S'ils ont été obtenus en supprimant deux facteurs disjoints, supprimer ces deux facteurs nous fournit la réduction z désirée.

Cette question est **difficile**, car elle requiert du candidat qu'il passe outre la simple intuition, selon laquelle la réduction complète de w est « évidemment unique », pour effectivement démontrer cette unicité. Le raisonnement décrit ci-dessus consiste en fait à démontrer la *confluence* d'un système de réécriture jouissant d'une priopriété de *diamant*. Il n'était évidemment pas attendu des candidats qu'ils soient familiers avec ces notions propres aux systèmes de réécriture, et que remplaçait une simple récurrence.

Question 4. Les mots réduits sont ceux qui évitent les facteurs $a\overline{a}$. L'ensemble $\mathsf{red_c}(\mathbb{A}^*)$ est donc un langage local, et il est rationnel.

Cette question a pour but de préparer le candidat aux questions plus difficiles qui suivent, et qui portent sur le caractère rationnel ou non contextuel de tel ou tel langage.

Les examinateurs ont été surpris de constater que seule une minorité de candidats a reconnu en $\mathsf{red_c}(\mathbb{A}^*)$ un langage local, l'écrasante majorité d'entre eux s'escrimant à construire de manière explicite un automate reconnaissant $\mathsf{red_c}(\mathbb{A}^*)$, un processus qui s'est souvent avéré laborieux.

Question 5. Le langage $\mathsf{red}^{-1}(L)$ n'est pas nécessairement rationnel, par exemple lorsque $L = \{\varepsilon\}$.

En effet, si $\mathsf{red}^{-1}(L)$ est rationnel, il satisfait le lemme de l'étoile : il existe un entier n tel que tout mot $w \in \mathsf{red}^{-1}(L)$ admet une factorisation $w = xyz$ pour laquelle $|x| \leqslant n$, $1 \leqslant |y| \leqslant n$ et $xy^*z \subseteq \mathsf{red}^{-1}(L)$. On choisit alors le mot $w = a_1^{2n}\overline{a_1}^{2n}$, qui appartient manifestement à $\mathsf{red}^{-1}(L)$, que l'on factorise comme ci-dessus. Puisque xy et un préfixe de w de longueur au plus $2n$, il existe des entiers $k \geqslant 0$ et $\ell \geqslant 1$ tels que $x = a_1^k$ et $y = a_1^\ell$. Mais alors $xz = a_1^{2n-\ell}\overline{a_1}^n$ ne se réduit pas en ε, en contradiction du lemme de l'étoile.

Ainsi, $\mathsf{red}^{-1}(\{\varepsilon\})$ est irrationnel.

Il s'agit là d'une première question peu évidente, qui vise à tester l'intuition et l'inventivité des candidats, ainsi que leur aptitude à utiliser de manière pertinente le lemme de l'étoile.

Question 6. La réponse est en fait positive. En effet, soit $\mathcal{G}$ une grammaire non contextuelle qui engendre L. On lui ajoute un nouveau symbole non terminal, disons Z, puis les règles suivantes :

- les règles $Z \to ZZ$, $Z \to \varepsilon$, et une règle $Z \to aZ\overline{a}$ pour chacun des $2n$ éléments de $\mathbb{A}$: cela permet à Z d'engendrer l'ensemble des mots qui se réduisent en ε ;
- pour tout symbole terminal X, les règles $X \to XZ$ et $X \to ZX$, qui permettent d'insérer en tout endroit de notre mot de départ un facteur réductible au mot vide.

Cette question a pour but de permettre au candidat de raisonner de manière simple sur les langages non contextuels et leurs grammaires.

Question 7. La réponse est positive. En effet, soit $\mathcal{A}$ un automate fini qui reconnaît L. Tant qu'il existe une paire d'états (p, q) reliés par un chemin d'étiquette $a\overline{a}$, on ajoute une ε-transition de p vers q (si une telle ε-transition n'existait pas déjà) : cela permet de simuler la suppression d'un tel facteur $a\overline{a}$.

On recommence ce processus tant que l'on peut ajouter des ε-transitions. Puisqu'il n'existe qu'un nombre fini de telles ε-transitions possibles, notre processus finira par prendre fin : on aura bien un automate qui contient les mots de plus petit langage contenant L et stable par suppression de facteurs $a\overline{a}$, c'est-à-dire $\mathsf{red}(L)$.

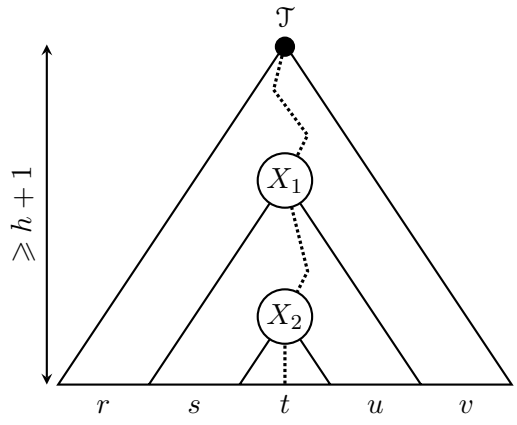
L'ensemble $\mathsf{red}_\mathsf{c}(L)$ est ensuite manifestement rationnel, en tant qu'intersection entre les langages rationnels $\mathsf{red}(L)$ et $\mathsf{red}_\mathsf{c}(\mathbb{A}^*)$.

Cette question a pour but de pousser à manipuler des automates de manière non triviale, ici en introduisant un critère (l'existence de chemins d'étiquette donnée) que l'on peut certes tester de manière algorithmique mais qu'il n'est pas nécessaire de savoir tester de manière algortihmique. Elle vise également à vérifier si le candidat parvient à utiliser les questions précédentes pour mettre en œuvre des raisonnements simples ; ici, se rappeler que $\mathsf{red}_\mathsf{c}(\mathbb{A}^*)$ est rationnel pour invoquer un simple argument d'intersection, plutôt que de tenter de construire explicitement un automate reconnaissant $\mathsf{red}_\mathsf{c}(L)$.

Question 8. Soit $\mathcal{G}$ une grammaire non contextuelle qui engendre L, et qui possède h symboles non terminaux. À partir de cette grammaire, si on se restreint à n'utiliser que des arbres de dérivation de hauteur au plus h, on ne peut produire qu'un nombre fini de mots ; soit n un entier strictement supérieur aux longueurs de ces mots.

On considère désormais un éventuel mot $w \in L$ de longueur $|w| \geqslant n$, puis un arbre de dérivation $\mathcal{T}$ permettant de construire le mot w, et choisi de manière à avoir le moins de nœuds possible. Par construction de n, $\mathcal{T}$ possède un chemin de hauteur $h + 1$, sur lequel un symbole non terminal, disons X, se retrouve au moins deux fois, en deux occurrences qu'on note X_1 (celle de hauteur maximale) et X_2 (celle de hauteur minimale). L'arbre $\mathcal{T}$ se dérive en le mot $w = rstuv$, où stu est le facteur de w dérivé du sous-arbre enraciné en X_1, et t est le facteur de w dérivé du sous-arbre enraciné en X_2.

Dès lors, on dispose de la règle de dérivation $X \to sXu$, donc chacun des mots rs^ktu^kv appartient à L. Comme $\mathcal{T}$ a été choisi minimal, les deux mots s et u ne sont pas tous deux nuls ; sinon, on aurait directement utilisé X_2 en lieu et place de l'occurrence X_1. Si $s \neq \varepsilon$, on a donc gagné ; sinon, on sait tout de même que $u \neq \varepsilon$, et il suffit alors de changer de variables, en considérant les facteurs $r_2 = rt$, $s_2 = u$, $t_2 = u_2 = \varepsilon$ et $v_2 = v$ au lieu de r, s, t, u et v.



Cette question a pour objet la démonstration d'une variante d'un lemme d'itération (aussi appelé lemme de pompage) pour un langage non contextuel. Ce lemme servira ensuite en question 9, pour laquelle l'inégalité $|stu| \leqslant n$ usuellement incluse dans les conclusions du lemme d'iteration n'est pas pertinente, et a donc été omise.

Question 9. Le langage L est engendré par la grammaire formée des règles $X \to \varepsilon$ et $X \to \varphi(a_i)X\overline{a_i}$. On entreprend désormais de démontrer que $\mathsf{red}(L)$ et $\mathsf{red}_{\mathsf{c}}(L)$ sont contextuels.

Pour ce faire, on utilise le lemme suivant : l'intersection d'un langage rationnel M et d'un langage non contextuel N est non contextuelle. En effet, si $\mathcal{A} = (Q, \mathbb{A}, \delta, \imath, F)$ est un automate qui reconnaît M et $\mathcal{G}$ est une grammaire non contextuelle (de symbole initial S) qui engendre N, on crée une nouvelle grammaire $\mathcal{G}'$ comme suit :

- pour plus de simplicité par la suite, on suppose que $\mathcal{G}$ ne consiste qu'en des règles $X \to a$ ou $X \to YZ$, où X, Y, Z sont non terminaux et a est une lettre de $\mathbb{A}$;
- pour chaque symbole non terminal X de $\mathcal{G}$ et chaque paire (p, q) d'états de $\mathcal{A}$, on ajoute à $\mathcal{G}'$ un symbole $X_{p\to q}$ non terminal, qui engendrera les mots qu'engendre X qui étiquettent un chemin de $\mathcal{A}$ allant de p à q ;
- pour chaque règle $X \to YZ$ de $\mathcal{G}$ et chaque triplet (p, q, r) d'états de $\mathcal{A}$, on ajoute à $\mathcal{G}'$ une règle $X_{p\to r} \to Y_{p\to q}Z_{q\to r}$;
- pour chaque règle $X \to a$ de $\mathcal{G}$ et chaque paire (p, q) d'états de $\mathcal{A}$ telle que $q \in \delta(p, a)$, on ajoute à $\mathcal{G}'$ une règle $X_{p\to q} \to a$.

De la sorte, les symboles $X_{p\to q}$ engendrent bien les langages désirés, et il suffit de prendre pour symboles initiaux de $\mathcal{G}'$ les symboles non terminaux $S_{\imath\to v}$ pour tout état $f \in F$.

Une fois ce lemme établi, on s'attache à démontrer que le langage $M = \mathsf{red}_{\mathsf{c}}(L) \cap a_1^+a_2^+a_3^+$ ne satisfait pas le lemme d'itération de la question 8. De la sorte, il sera contextuel, et $\mathsf{red}(L)$ et $\mathsf{red}_{\mathsf{c}}(L)$ aussi.

Un mot $x \in \Sigma^*$ appartient à $\mathsf{red}_{\mathsf{c}}(L)$ si et seulement s'il existe un mot w tel que $xw = \varphi(w)$. Si x et non vide, $|w| = |x|/3$ est non vide aussi : c'est un suffixe strict de $\varphi(w)$. En particulier, tout suffixe s de w est un suffixe de $\varphi(s)$.

C'est par exemple le cas de la dernière lettre de w, qui est donc nécessairement a_4. Une récurrence sur k indique alors qu'il existe un entier $k \geqslant 0$ tel que $\varphi^k(a_4)$ est un suffixe de w, qui est lui-même un suffixe de $\varphi^{k+1}(a_4)$. Mais alors xw est un suffixe de $\varphi^{k+2}(a_4)$.

Or, si l'on note $\mathbf{A}(k)$ le mot $a_{k+1}^{4^k}a_{k+2}^{4^k}a_{k+3}^{4^k}$, où les indices sont pris modulo 3, une récurrence sur k indique que $\varphi^{k+1}(a_4) = \mathbf{A}(k)\mathbf{A}(k-1)\cdots\mathbf{A}(0)a_4$. Par conséquent, tout mot $x \in M$ est facteur d'un mot $\mathbf{A}(3k)$, et il est associé à un mot w donc $\mathbf{A}(3k-1)\mathbf{A}(3k-1)\cdots\mathbf{A}(0)a_4$ est un suffixe. Mais alors $3 \times 4^{3k} \geqslant |x|$ et $|w| \geqslant 4^{3k}$, donc l'inégalité $|x| = 3|w|$ n'est possible que si $x = \mathbf{A}(3k)$ et $w = \mathbf{A}(3k-1)\cdots\mathbf{A}(0)a_4$, auquel cas l'égalité $xw = \varphi(w)$ est bien satisfaite.

En conclusion, le langage M est donc constitué des mots $\mathbf{A}(3k)$, c'est-à-dire de mots de longueur 3×4^{3k}. Or, le lemme d'itération indique que les longueurs de tout langage non contextuel infini contiennent des suites arithmétiques. Ce n'est pas le cas de M, qui est donc contextuel.

Cette question **très difficile** a pour but d'occuper les meilleurs candidats, qui ont de toute façon déjà obtenu la note de 20/20 s'ils font quelque remarque pertinente que ce soit sur cette question.