Calculabilité / Complexité (L3)
Examen "Complexité"
Énoncés et solutions*

Exercice (6 points)

Pour un entier $k > 0$ et un alphabet fini A, une fonction totale $f : A^* \to A^*$ est dite $\log^k$space s'il existe une machine de Turing déterministe qui calcule $f(x)$ pour tout $x \in A^*$ en utilisant un espace de travail $O(\log^k n)$, c.-à-d. $\leq c(\log |x|)^k$, pour un entier c. On dit que f est "logspace" quand $k = 1$, et polylogspace si elle est $\log^k$space pour un $k \in \mathbb{N}$.

Note: pour simplifier les calculs, on parle ici d'une fonction $\log : \mathbb{N} \to \mathbb{N}$ définie par $\log(0) = 0$ et, pour $n > 0$, $\log(n) = \lfloor \log_2 n \rfloor$. Dans le même esprit, on convient que l'adjectif "logspace" et ses dérivés sont invariables.

Question 1. Est-ce que les fonctions polylogspace sont calculables en temps polynomial? Justifiez.

Solution. Non. Une machine de Turing déterministe peut calculer $\log^k n$ et compter jusqu'à $2^{\log^k n}$ en utilisant une mémoire de travail en $O(\log^k n)$. Elle peut donc calculer la fonction $\mathtt{I}^n \mapsto \mathtt{I}^{2^{\log^k n}}$ sur l'alphabet $A = \{\mathtt{I}, ..\}$. Or le temps de calcul est minoré par la longueur du résultat (qu'il faut bien écrire sur le ruban de sortie) et $2^{\log^k n} = n^{\log^{k-1} n}$ ne peut pas être dominé par un polynôme si $k > 1$. (Autre réponse possible: invoquer le "*Space Hierarchy Theorem*", c.-à-d. le Thm 10 du cours.)

Question 2. Montrez que si f est $\log^k$space alors $|f(x)|$ est en $|x|^{O(\log^{k-1} |x|)}$.

Solution. Soit M une machine qui calcule f. Notons $n = |x|$. À tout instant M est dans une configuration de la forme (q, i, w, j, o) où q est un état de contrôle, i une position de la tête de lecture sur l'entrée x, w un contenu du ruban de travail, j une position de tête de lecture sur w, et o le contenu actuel du ruban de sortie. Puisque M est $\log^k$space, il y a $2^{O(\log^k |x|)}$ contenus w possibles, et donc —si on ne tient pas compte de o, qui n'influence pas le comportement— au plus $|Q| \times (n + 2) \times 2^{O(\log^k n)} \times O(\log^k n)$ configurations distinctes possibles. On a donc $2^{O(\log^k n)}$ configurations possibles et, puisque la machine M est déterministe, elle ne peut pas visiter deux fois la même configuration (sinon elle bouclerait). Donc M termine en temps $2^{O(\log^k n)}$ ce qui borne $|f(x)|$.

Question 3. Est-ce que la composition de deux fonctions $\log^k$space est elle-même $\log^k$space? Justifiez.

Solution. Non. On a vu que la fonction $f : \mathtt{I}^n \mapsto \mathtt{I}^{2^{\log^2 n}}$ est $\log^2$space. Or l'image de $\mathtt{I}^n$ par $f \circ f$ est de longueur $2^{\log^2 m}$ pour $m = 2^{\log^2 n}$, c.-à-d. de longueur $2^{(\log^2 n)^2}$, ou $2^{\log^4 n}$. On sait donc par la question précédente que $f \circ f$ n'est pas $\log^2$space. (On peut aussi invoquer le Thm 10.)

Problème (14 points)

On dit qu'un mot u est sous-mot d'un mot v, noté $u \preceq v$, si u est une sous-suite de v, c.-à-d., obtenue en retirant un nombre arbitraire de lettres. P.ex. abba $\preceq$ abracadabra. En particulier, $\epsilon \preceq u \preceq u$ pour tout u, où ϵ dénote le mot vide.

On s'intéresse au problème `LCS` (pour "Longest Common Subword"). Il prend en entrée un alphabet A, une liste $u_1, \ldots, u_k$ de mots dans A^*, ainsi qu'un entier N. La question à résoudre est "existe-t-il un mot v tel que $|v| = N$ et $v \preceq u_i$ pour $i = 1, \ldots, k$?"

*Merci de signaler toute erreur ou typo à `phs@lsv.fr`.

Question 1. Est-ce que l'instance $(A = \{a, b, c\}, u_1 = aabc, u_2 = bbca, u_3 = ccab, N = 2)$ est positive? Est-ce que l'instance

$$(A = \{0, 1, \ldots, 9\}, u_1 = 314159265358979323846, u_2 = 141421356237309504880, N = 8)$$

est positive?

Justifiez brièvement dans les deux cas.

Solution. Première question : non. Deuxième question: oui, il y a même un sous-mot commun de longueur 9.

Question 2. On suppose que dans une instance de LCS, l'entier N est donné en base 1, p.ex. sous la forme $\mathtt{I}^N = \mathtt{II}\ldots\mathtt{I}$, et donc une instance a une longueur $O(N + L \log L)$ si $L = |u_1| + \cdots + |u_k|$. (NB: le facteur $\log L$ compense le fait que les u_i peuvent utiliser jusqu'à L lettres différentes.)

Montrez que LCS est dans NP.

Solution. Un algorithme NP possible consiste à deviner le sous-mot commun de longueur N et à le valider en le comparant aux u_is. Pour tester $v \preceq u_i$ on peut utiliser un algorithme "glouton" qui calcule le plongement le plus à gauche et répond en temps $O(|u_i|)$, mais on peut aussi deviner une façon dont v se plonge dans u_i (et ensuite la valider) puisque un algorithme non déterministe nous suffit.

Question 3. Soit LCS_b le problème qui est comme LCS sauf que N est écrit en base 2, de sorte que la taille d'une instance est $O(\log N + L \log L)$.

3a. Donnez une réduction logspace de LCS à LCS_b.

3b. Donnez ensuite une réduction logspace de LCS_b à LCS .

Pour ces deux questions on justifiera la correction et on détaillera (sans forcément écrire un programme) les algorithmes utilisés par les réductions de façon à bien comprendre comment un espace logarithmique est suffisant.

Solution. Pour réduire LCS à LCS_b la seule difficulté est de traduire une suite de N bâtons en une écriture binaire de N. On maintient un compteur binaire en mémoire de travail et on l'incrémente (en base deux) à chaque fois qu'on lit un bâton. À la fin on l'écrit sur la sortie. L'espace de travail est en $\log N$ donc en $\log n$. Réduire LCS_b à LCS est plus délicat: un algorithme logspace ou même ptime ne peut pas produire $\mathtt{I}^N$ à partir d'une écriture binaire de N car cela prendrait un temps $\Omega(N)$ qui peut être exponentiel en n. Mais une réduction qui transforme $u_1, \ldots, u_k, N$ en $u_1, \ldots, u_k, \mathtt{I}^{\min(L+1,N)}$ est correcte car les u_i ne peuvent pas avoir de sous-mot commun de longueur $> L$. Pour implémenter cette réduction il faut calculer $L + 1$ en maintenant un compteur binaire en mémoire de travail, calculer $N' = \min(L + 1, N)$ en comparant deux entiers écrits en binaire. Le calcul de N' tient en espace $O(\log n)$ et on peut produire $\mathtt{I}^{N'}$ p.ex. par une boucle qui décrémente N' et produit un I jusqu'à atteindre $N' = 0$. (Autre possibilité: si $N \leq L$ produire $u_1, \ldots, u_k, \mathtt{I}^N$, sinon produire une instance trivialement négative comme $\epsilon, \mathtt{I}$.)

Question 4. On veut montrer que LCS est NP–difficile. Pour cela on part du problème NODECOVER vu en TD et connu pour être NP-complet. On rappelle qu'une instance de NODECOVER est constituée d'un graphe simple (i.e., non orienté, sans arêtes multiples ni boucles) $G = (V, E)$ et d'un entier K et qu'on se demande si G admet un recouvrement de cardinal au plus K, sachant qu'un recouvrement est un ensemble de sommets $C \subseteq V$ tel que chaque arête de E a une (au moins) de ses extrémités dans C.

Soit une instance $G = (V, E), K$ avec $|V| = \ell$ sommets et $|E| = m$. On va considérer des mots $u_0, u_1, \ldots, u_m$ sur l'alphabet V. On pose d'abord $u_0 = v_1 v_2 \ldots v_l$ en fixant une énumération de V. On fixe ensuite une énumération $e_1, e_2, \ldots, e_m$ des arêtes et pour, $i \in \{1, \ldots, m\}$, on pose $e_i = \{v_r, v_s\}$ de sorte que $r < s$. On pose alors

$$u_i = v_1 v_2 \ldots v_{r-1} v_{r+1} \ldots v_l v_1 v_2 \ldots v_{s-1} v_{s+1} \ldots v_l$$

4a. Prouvez que G admet un recouvrement de taille $\ell - N$ ssi il existe un mot w de longueur N qui soit sous-mot de chacun des u_i pour $i = 0, \ldots, m$.

Solution. Supposons que w existe. C'est un sous-mot de u_0 donc une suite $v_{i_1} \dots v_{i_N}$ de sommets avec $1 \leq i_1 < i_2 < \cdots < i_N \leq \ell$. On montre que $C = V \setminus \{i_1, \dots, i_\ell\}$ est un recouvrement. Prenons une arête $e_i = \{v_r, v_s\}$ de E avec $r < s$. Le mot u_i ne contient qu'une occurrence de v_r, qu'une occurrence de v_s, et v_r apparaît à droite de v_s. Donc w ne peut pas contenir à la fois v_r et v_s. Donc C contient v_r ou v_s. Puisque ceci vaut pour toute arête, C est bien un recouvrement. Enfin $|C| = \ell - N$ car un sous-mot de u_0 n'a pas de sommets en double.
Réciproquement si G admet un recouvrement C de taille $\ell - N$ alors le mot $w = v_{i_1} \dots v_{i_N}$ obtenu en énumérant dans l'ordre croissant les sommets de $V \setminus C$ est sous-mot de chacun des u_i. C'est évident pour u_0 qui contient tous les sommets en ordre croissant. Pour $i \in \{1, \dots, m\}$ on pose $e_i = \{v_r, v_s\}$ et on remarque que, puisque C est un recouvrement, un sommet au moins parmi v_r et v_s est dans C donc ne figure pas dans w. Si c'est v_r alors w est sous-mot de la première moitié de u_i (de la deuxième moitié si c'est v_s).

4b. Donnez une réduction logspace de NODECOVER à LCS. Justifiez soigneusement (sans écrire de code) que votre réduction est bien calculable en espace logarithmique.

Solution. La réduction est presque complètement donnée à la question précédente sauf qu'il faut préciser $N = \ell - K$.
Pour expliquer la calculabilité en logspace, on supposera que G est donné sous la forme $v_1, \dots, v_\ell, \{v_{s_1}, v_{r_1}\}, \dots, \{v_{s_m}, v_{r_m}\}, K$ de sorte que produire u_0 est trivial. Pour produire les u_i's il faut maintenir un compteur pour i, identifier v_{s_i} et v_{r_i} dans $v_1, \dots, v_\ell$ de façon à pouvoir mémoriser s_i et r_i (deux compteurs). On peut ensuite produire u_i puis passer à $i+1$. Trois compteurs binaires sont ainsi utilisés. Enfin on calcule $\ell - K$ de façon logspace.

4c. Conclure en donnant la complexité de LCS et celle de LCS_b.

Solution. Au vu des questions précédentes, les deux problèmes sont NP-complets.

Question 5. On s'intéresse à deux versions de LCS. Pour deux mots u, v, on note $u \preceq_{\text{no}} v$, et on dit que "u est un sous-mot non orienté de v", ssi $u \preceq v$ ou $\widetilde{u} \preceq v$. Ici $\widetilde{u}$ est le mot miroir de u: p.ex. $\widetilde{\texttt{aab}} = \texttt{baa}$. De même on note $u \preceq_{\text{cy}} v$, et on dit que "u est cycliquement sous-mot de v", si $u_2u_1 \preceq v$ pour une factorisation $u = u_1u_2$ de u.

5a. Est-ce que $\preceq_{\text{no}}$ est un préordre, c.-à-d., une relation réflexive et transitive, sur les mots? Même question pour $\preceq_{\text{cy}}$. Justifiez.

Solution. Oui dans les deux cas. Pour la transitivité de $\preceq_{\text{no}}$ on utilise $\widetilde{u} \preceq v$ ssi $u \preceq \widetilde{v}$.
Pour celle de $\preceq_{\text{cy}}$ on peut noter que $\preceq_{\text{cy}}$ est défini comme $\preceq \circ \sim$, où $\sim$ est la relation de conjugaison: $uu' \sim u'u$. Il suffit alors de montrer que $\preceq \circ \sim$ et $\sim \circ \preceq$ coïncident et d'invoquer la transitivité de $\preceq$ et de $\sim$.

5b. Le problème LCS_{no} est une version de LCS où on demande s'il existe un sous-mot non orienté v de longueur N tel que $v \preceq_{\text{no}} u_i$ pour $i = 1, \dots, k$?

Montrez que LCS_{no} est NP-complet.

Solution. Le problème est évidemment dans NP: on devine v et tester $v \preceq_{\text{no}} u_i$ reste faisable en temps polynômial (idem pour $v \preceq_{\text{cy}} u_i$). Pour montrer la NP-difficulté on peut réduire LCS à LCS_{no} via

$$(A, u_1, \dots, u_k, N) \mapsto (A \cup \{\$, £\}, \$^{L+1}u_1£^{L+1}, \dots, \$^{L+1}u_k£^{L+1}, N + 2L + 2)$$

où $L = \max |u_i|$ et où $\$, £$ sont deux symboles nouveaux n'apparaissant pas dans A. (Justifications de la correction omises).

5c. Même question pour le problème LCS_{cy} qui demande si les u_i admettent un sous-mot commun de longueur N au sens de $\preceq_{\text{cy}}$.

Solution. La même réduction marche pour montrer la NP-difficulté.

Question 6. On note LCS_2 la restriction de LCS où les instances contiennent exactement deux mots, i.e., $k = 2$.

6a. Donnez un algorithme en PTIME qui résout LCS_2. Justifiez sa correction et votre analyse de complexité.

Solution. On peut résoudre LCS_2 en temps quadratique par une technique de programmation dynamique. Définissons $plsmc : A^* \times A^* \to A^*$ via $plsmc(\epsilon, v) = plsmc(u, \epsilon) = \epsilon$ et, pour deux mots non vides, $plsmc(a.u, a'.v) = a.plsmc(u, v)$ si $a = a'$, et = "le plus long parmi $plsmc(u, a'.v)$ et $plsmc(a.u, v)$" si $a \neq a'$. Alors $plsmc(u, v)$ est un plus long sous-mot commun à u et v (justification omise). Vous pouvez programmer cet algorithme et le tester sur la question 1.

6b. Plus généralement, pour $k \in \mathbb{N}$ fixé, le problème LCS_k est la restriction de LCS où les instances contiennent exactement k mots $u_1, \ldots, u_k$.

Soit $k \in \mathbb{N}$. Est-ce que LCS_k est dans PTIME ?

Solution. La technique précédente se généralise pour donner un algorithme en temps $O(n^k)$. Donc LCS_k est dans PTIME.