Examen du cours Complexité (L3)

Les documents (notes, polycopiés, ..) et calculatrices (téléphone, tablette, ..) ne sont pas autorisés.

Date : 13 janv. 2025 à 10h45 / Durée : 2 heures

Exercice : Fonctions polylogspace

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 discrète $\log : \mathbb{N} \to \mathbb{N}$ définie par $\log(0) = 0$ et, pour $n > 0$, $\log(n) = \lfloor \log_2 n \rfloor$. Toujours pour simplifier, on convient que l'adjectif *logspace* et ses dérivés sont invariables.

1. Donnez une fonction polylogspace qui n'est pas calculable en temps polynomial. Justifiez.

   Solution:
   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 cette longueur, c.-à-d. $2^{\log^k n} = n^{\log^{k-1} n}$, ne peut pas être dominée par un polynôme si $k > 1$.
   Commentaire : le problème d'accessibilité dans les graphes orientés, c.-à-d. GAP, n'est pas une réponse valide. Il est bien connu, et a néanmoins été explicitement rappelé en cours, que ce problème est PTIME.

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) 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 (qu'on suppose unique pour simplifier), j une position de tête de lecture sur w. Puisque M est $\log^k$space, il y a $2^{O(\log^k |x|)}$ contenus w possibles, et donc 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 et termine toujours, elle ne peut pas visiter deux fois la même configuration. Donc M termine en temps $2^{O(\log^k n)}$ ce qui borne $|f(x)|$.

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 du polycopié.)
   Commentaire : Plusieurs copies ont cru pouvoir adapter la composition des réductions logspace (vue en cours) pour en faire une composition de réductions $\log^k$space. Mais pour le calcul de $g \circ f$ avec f et g

$\log^k$space, la simulation de g doit traiter un input de longueur $|f(x)|$ qui est $O(n^{\log^{k-1} n})$, donc non borné polynomialement, et utilisera donc un espace de travail en $O(\log^k(|f(x)|))$ qui ne sera donc pas en $O(\log^k n)$.

Problème : Sous-mots et sous-suites

On dit qu'un mot u est sous-mot d'un mot v, noté $u \preccurlyeq v$, si u est une sous-suite de v, c.-à-d., obtenue en retirant un nombre arbitraire de lettres. P.ex. `examen` $\preccurlyeq$ `inexorablement` et `examen` $\not\preccurlyeq$ `existentialisme`. En particulier, $\epsilon \preccurlyeq u \preccurlyeq u$ pour tout u, où ϵ dénote le mot vide.

On s'intéresse au problème LCS (pour "Long Common Subword"). Le problème LCS 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 $v \in A^*$ tel que $|v| = N$ et $v \preccurlyeq u_i$ pour $i = 1, \ldots, k$? »

4. Parmi les deux instances suivantes

$$(A = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\},\ u_1 = \mathtt{aabc},\ u_2 = \mathtt{bbca}, u_3 = \mathtt{ccab},\ N = 2)\,,$$
$$(A = \{\mathtt{0}, \mathtt{1}, \ldots, \mathtt{9}\},\ u_1 = \mathtt{31415926535897932384},\ u_2 = \mathtt{14142135623730950488},\ N = 8)\,,$$

dites laquelle est positive et justifiez brièvement.

Solution:
La deuxième. Il y a même des sous-mots communs de longueur 9 comme `141563794`.

On précise que le problème LCS est un langage sur un alphabet Σ fixé. Donc la donnée d'un alphabet A (qui peut être n'importe quel alphabet fini) est codée sur Σ et la donnée des u_i reprend ce codage. Par ailleurs l'entier N est donné en base 1, p.ex. sous la forme $\mathtt{I}^N = \mathtt{II}\cdots\mathtt{I}$ avec $\mathtt{I} \in \Sigma$. En fin de compte, la taille d'une instance sera en $O\big(k+N+(|A|+|u_1|+\cdots+|u_k|)\log|A|\big)$ où le facteur multiplicatif $\log|A|$ rend compte du codage des lettres de A sur l'alphabet Σ.

5. Montrez que LCS est dans NP.

Solution:
Un algorithme NP possible consiste à deviner le sous-mot commun de longueur N, à le stocker sur un ruban de travail, et à le valider en le comparant aux différents u_i's. Pour tester $v \preccurlyeq u_i$ on peut utiliser un algorithme qui calcule le plongement le plus à gauche et répond en temps $O(|u_i|)$, mais on peut aussi deviner, pour chaque $i = 1, \ldots, k$, un sous-mot de u_i et le comparer à v puisque un algorithme non déterministe nous suffit.

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 en $O(k + \log N + (|A| + |u_1| + \cdots + |u_k|)\log|A|)$.

6. Donnez une réduction logspace de LCS à LCS_b.
   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, en notant $L = |u_1| + \cdots + |u_k|$, il suffit que notre réduction transforme N (écrit en binaire) en $\mathtt{I}^{N'}$ avec $N' = \min(L+1, N)$ car les u_is ne peuvent pas avoir de sous-mot commun de longueur $> L$. (On peut aussi prendre $L = \max_i |u_i|$ mais c'est plus compliqué à calculer que $\sum_i |u_i|$.) Pour implémenter cette réduction on calcule d'abord la représentation binaire de $L+1$ en incrémentant un compteur binaire en mémoire de travail tandis qu'on parcourt l'instance. Ensuite on calcule $N' = \min(L+1, N)$ 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 chaque fois produit un $\mathtt{I}$ 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 $\epsilon, \mathtt{I}$.)

Commentaire : La majorité des copies ont cru expliquer comment on pourrait générer l'écriture unaire d'un nombre binaire en espace logarithmique, *et donc en temps polynomial!!*. Une pratique habituelle de la programmation et les réflexes algorithmiques basiques qui en découlent devraient normalement interdire ce genre d'erreurs.

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 au moins une de ses extrémités dans C.

Soit une instance $G = (V, E), K$ avec $|V| = \ell$ sommets et $|E| = m$ arêtes. 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

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

7. Prouvez que G admet un recouvrement de taille $\ell - N$ si et seulement si 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 $v_{i_1} \ldots v_{i_N}$ de sommets avec $1 \leq i_1 < i_2 < \cdots < i_N \leq \ell$. On montre que $C = V \setminus \{i_1, \ldots, i_N\}$ 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 $w = v_{i_1} \ldots v_{i_N}$ 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 $i \in \{1, \ldots, m\}$ 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 (et de la deuxième moitié si c'est v_s).

8. 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 l'alphabet A et donner $N = \ell - K$.
Pour expliquer la calculabilité en logspace, on supposera que G est donné sous la forme $v_1, \ldots, v_\ell, \{v_{s_1}, v_{r_1}\}, \ldots, \{v_{s_m}, v_{r_m}\}, K$ de sorte que produire u_0 est trivial. Pour produire

les u_is il faut maintenir un compteur pour i, identifier v_{s_i} et v_{r_i} dans $v_1, \ldots, v_\ell$ 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.

9. Conclure en donnant la complexité de LCS et celle de LCS_b.

Solution:
Au vu de la question 8, LCS est NP-difficile. Avec la question 5, il est NP-complet. Avec la question 6, c'est aussi le cas de LCS_b.

On s'intéresse à deux versions de LCS. Pour deux mots u, v, on note $u \preccurlyeq_{\text{no}} v$, et on dit que « u est un sous-mot non orienté de v », ssi $u \preccurlyeq v$ ou $\widetilde{u} \preccurlyeq v$. Ici $\widetilde{u}$ le mot miroir de u : p.ex. $\widetilde{\texttt{engager}} = \texttt{regagne}$. De même on note $u \preccurlyeq_{\text{cy}} v$, et on dit que « u est cycliquement sous-mot de v », si $u_2u_1 \preccurlyeq v$ pour une factorisation $u = u_1u_2$ de u. P.ex., $\texttt{avril} \preccurlyeq_{\text{cy}} \texttt{invraisemblable}$ puisque $\texttt{avril}$ est sous-mot de $\texttt{able} \cdot \texttt{invraisembl}$.

10. Montrez que $\preccurlyeq_{\text{no}}$ et $\preccurlyeq_{\text{cy}}$ sont des préordres, c.-à-d., des relations réflexives et transitives, sur les mots ?

Solution:
La réflexivité est triviale. Pour la transitivité de $\preccurlyeq_{\text{no}}$ on utilise $\widetilde{u} \preccurlyeq v$ ssi $u \preccurlyeq \widetilde{v}$.
Pour celle de $\preccurlyeq_{\text{cy}}$ on note que $\preccurlyeq_{\text{cy}}$ est défini comme $\preccurlyeq \circ \sim$, où $\sim$ est la relation de conjugaison : $uu' \sim u'u$. On montre ensuite que $\preccurlyeq \circ \sim$ et $\sim \circ \preccurlyeq$ coïncident, comme suggéré par l'exemple : puisque $\texttt{avril} \preccurlyeq \texttt{able} \cdot \texttt{invraisembl}$, alors $\texttt{avril}$ a un conjugué $\texttt{vril} \cdot \texttt{a}$ qui est sous-mot de $\texttt{invraisemblable}$. On peut donc écrire

$$\preccurlyeq \circ \sim \;=\; \sim \circ \preccurlyeq \circ \sim \;=\; \sim \circ \preccurlyeq$$

et en déduire (en utilisant la transitivité de $\preccurlyeq$)

$$\preccurlyeq_{\text{cy}}^2 \;=\; (\preccurlyeq \circ \sim)^2 \;= \preccurlyeq \circ \sim \circ \preccurlyeq \circ \sim \;=\; \sim \circ \preccurlyeq \circ \preccurlyeq \circ \sim \;=\; \sim \circ \preccurlyeq \circ \sim \;=\; \preccurlyeq_{\text{cy}} \;.$$

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 $v \preccurlyeq_{\text{no}} u_i$ pour $i = 1, \ldots, k$?

11. Montrez que LCS_{no} est NP-complet.

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

$$(A, u_1, \ldots, u_k, N) \mapsto (A \cup \{\mathtt{A}, \mathtt{B}\}, \mathtt{A}^{L+1}u_1\mathtt{B}^{L+1}, \ldots, \mathtt{A}^{L+1}u_k\mathtt{B}^{L+1}, N + 2L + 2)$$

où $L = \max |u_i|$ et où $\mathtt{A}, \mathtt{B}$ sont deux symboles nouveaux n'apparaissant pas dans A. (Justifications de la correction omises).
Commentaire : Plusieurs copies ont proposé la réduction $(A, u_1, \ldots, u_k, N) \mapsto (A, u_1\widetilde{u_1}, \ldots, u_k\widetilde{u_k}, 2N)$ et ont prétendu prouver sa correction. Observons alors que $\texttt{aabc}\,\texttt{cbaa}$ et $\texttt{ccba}\,\texttt{abcc}$ ont un sous-mot commun de longueur 5 ! !

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

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

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

13. 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 4.

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

14. 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, pour chaque valeur fixée de k, le problème LCS_k est dans PTIME.