Complexité - TD 06

Benjamin Bordais

07 Décembre 2021

Exercise 1 Quelques problèmes NL-complet

Montrer que les problèmes suivants sont NL-complet :

1. Décider si un automate non déterministe (sans ϵ-transition) accepte un mot w.
2. Décider si un graphe orienté est fortement connecté.
3. Décider si un graphe orienté a un cycle.

Solution :

Les preuves de NL-dureté se feront toutes avec une réduction à partir du problème d'accessibilité dans un graphe GAP.

1. Le problème est dans NL car on peut simplement deviner un chemin dans l'automate correspondant au mot considéré arrivant dans un état acceptant. Pour être en espace logarithmique, on ne devine pas le chemin en entier mais plutôt transition par transition en conservant un pointeur sur l'état courant et un compteur sur le nombre d'états déjà vus.

   Pour ce qui est de la NL-dureté, étant donné un graphe G, un nœud de départ s et d'arrivée t, on étiquette toutes les arêtes avec une lettre a quelconque pour créer un automate avec un état initial s et final t, avec une boucle sur l'état final étiqueté par la lettre considérée a. Alors, $(G, s, t) \in \mathsf{GAP}$ est équivalent au fait que $\mathcal{A}$ accepte le mot $w = a^n$, où n est le nombre d'état du graphe/automate.

2. Le problème est dans $\mathsf{coNL} = \mathsf{NL}$ car le complémentaire de ce langage est dans NL : on peut simplement deviner une paire de nœuds et vérifier qu'il ne sont pas connectés.

   Pour ce qui est de la dureté, étant donné un triplet (G, s, t), on construit G' à partir de G en rajoutant des arêtes de t vers tous les nœuds et de tous les nœuds vers s. Alors, G' est fortement connecté si et seulement si il y a un chemin de s vers t dans G'. La réduction peut bien s'effectuer en espace logarithmique.

3. Le problème est dans NL : à partir d'un graphe G, on devine une arête (x, y) du cycle et on lance un algorithme pour GAP sur (G, y, x).

   Considérons à présent la NL-dureté. A partir d'une instance (G, s, t) de GAP, on peut créer un graphe G' en ajoutant tout d'abord une arête entre t et s, créant ainsi un cycle dans G' si s et t sont connectés dans G. Cependant, cela ne suffit pas car G pourrait avoir d'autres cycles et l'équivalence ne tiendrait alors pas. Ainsi, on élimine préalablement tous les cycles de G. Soit m le nombre de nœuds de G. On crée alors m copies de G (qui peuvent être vus comme m niveaux consécutifs). Pour chaque arête de i vers j de G, on ajoute une arête du nœud i à chaque niveau vers le nœud j au niveau suivant. On appelle également s' le nœud s du premier

niveau et t' le nœud du dernier. On ajoute finalement une arête de chaque nœud t à un niveau quelconque vers le nœud t'. De cette manière, il y a un chemin de s vers t dans G si et seulement si il y en a un de s' vers t' dans G'. De plus, dans G', les chemins ne peuvent que "monter" en niveau, et il ne peut donc y avoir de cycle. On ajoute donc une arête de t' vers s' et on a alors qu'il y a un chemin de s vers t dans G si et seulement si il y a un cycle dans G'. On peut vérifier que cette opération peut se faire en espace logarithmique.

Exercise 2 Défnition alternative de NL

Une machine de Turing avec *certificat*, appelé un vérificateur, est une machine de Turing déterministe avec une bande additionnelle en lecture seule appelés la *bande de certificat*, qui est en outre, en *lecture unique*, c'est à dire que la tête de lecture ne peut pas faire de mouvement vers la gauche sur cette bande. Un vérificateur prend en entrée un mot x, avec en plus un mot écrit u écrit sur la bande de certificat.

On définit alors NL_{certif} comme la classe des langages L tels qu'il existe un polynôme $p : \mathbb{N} \to \mathbb{N}$ et un vérificateur V s'exécutant en espace logarithmique tel que :

$$x \in L \text{ ssi } \exists u, |u| \leq p(|x|) \text{ et } V \text{ accepte sur l'entrée } (x, u)$$

1. Montrer que $\mathsf{NL}_{certif} = \mathsf{NL}$
2. Quelle classe de complexité obtient-on si on enlève la contrainte de lecture unique sur la bande de certificat ?

Solution :

1. $\mathsf{NL}_{certif} \subseteq \mathsf{NL}$: Soit A un langage de NL_{certif}. Si V est un vérificateur pour A, on peut simuler l'exécution de V sur une entrée x en devinant la lettre suivante de u en acceptant x si et seulement si V aurait accepté sur (x, u) en gardant un compteur pour vérifier que le calcul n'aurait pas nécéssité un u plus grand que $p_M(|x|)$. Cela s'effectue en espace logarithmique. En effet, V est en espace logarithmique et on a seulement besoin de stocker une lettre de u à la fois, étant donné que le certificat est en lecture seule.
   $\mathsf{NL} \subseteq \mathsf{NL}_{certif}$: Soit à présent A dans NL, et M machine de Turing non déterministe en espace logarithmique reconnaissant A. Pour une entrée x, une exécution de M sur x peut être caractérisée par tous les choix non déterministes dans son arbre d'exécution, ce qui peut être encodé par un mot y sur l'alphabet $\{0, 1\}^*$ avec $|y| \leq p_M(|x|)$ pour une fonction polynomiale p_M bornée par le temps d'exécution de M (qui existe puisque M s'exécute en espace logarithmique donc en temps polynomial). Il faut remarquer que cette fonction polynomiale ne dépend pas de l'entrée mais seulement de M. On peut alors construire un vérificateur V qui simule M et consulte la bande de certificat chaque fois que M utilise du non-déterminisme. L'exécution de V est déterministe et est en espace logarithmique.
2. On appelle cette nouvelle classe NL'_{certif}.
   $\mathsf{NL}'_{certif} \subseteq \mathsf{NP}$: Soit A dans NL'_{certif} et considérons et vérificateur V pour A. On construit une machine de Turing non-déterministe fonctionnant en temps polynomial acceptant A. D'abord, M devine le certificat (qui est polynomial en la taille de l'entrée), et stocke sur une bande de travail vu qu'elle peut être lu plusieurs fois. A présent, M exécute V sur (x, u), ce qui prend un temps polynomial étant donné que V fonctionne en temps polynomial.

$\mathsf{NP} \subseteq \mathsf{NL}'_{certif}$: D'abord, montrons que $\mathsf{SAT} \in \mathsf{NL}'_{certif}$. Considérons une formule propositionnelle φ. On csontruit un vérificateur V qui vérifie si le certificat sur la bande de certificat qui donne une valuation booléenne $(a_1, \ldots, a_n)$ des variables satisfait la formule φ. Spécifiquement, V boucle sur toutes les clauses et consulte les valuations des littéraux correspondant à la recherche d'un satisfaisant la clause. Cela peut s'effectuer en espace logarithmique vu que l'on a seulement besoin d'un pointeur sur la clause et le littéral d'intérêt. Remarquer que la bande de certificat est consulté pour chaque clause, ce qui ne serait pas possible en lecture seule. Il faut ensuite montrer que NL'_{certif} est clos par réduction en espace logarithmique. Cela se fait de manière analogue à la preuve que la relation entre langage "peut être réduits en espace logarithmique" est transitive.

On conclut finalement que $\mathsf{NP} \subseteq \mathsf{NL}'_{certif}$.

Exercise 3 Langages réguliers

Notons REG l'ensemble des langages réguliers/rationnels.

1. Montrer que, pour tout $L \in \mathsf{REG}$, L est reconnu par une TM s'exécutant en espace 0 et en temps $n + 1$.
2. Exhiber un langage reconnu par une machine de Turing en espace $O(\log n)$ et en temps $O(n)$ qui n'est pas dans REG.

Solution :

1. Soit $L \in \mathsf{REG}$. Ce langage est reconnu par un automate fini $\mathcal{A}$. On considère alors la machine de Turing avec les même états que $\mathcal{A}$ qui, sur une entrée w, simule l'exécution de w sur $\mathcal{A}$ et accepte si $\mathcal{A}$ accepte. Cette machine de Turing ne prend aucun espace et s'exécute en temps $n + 1$ (la $n + 1$-ième étape lit le premier blanc après l'entrée et accepte/rejette).
2. Le langage $L = \{a^n \cdot b^n \mid n \geq 0\}$ n'est pas régulier et peut être reconnu par une machine de Turing qui compte le nombre de a avec un compteur en binaire, le décrémente par chaque b vu et accepte si, à la fin du mot, le compteur est à 0.

Exercise 4 Too fast !

Montrer que $\mathsf{NTIME}(\log\ n) \neq \mathsf{L}$.

Solution :

Lorque l'on considère $\mathsf{NTIME}(\log\ n)$, on n'a même pas le temps de lire l'entrée en intégralité. Ainsi, tout langage qui est dans L et qui nécessite que l'entrée soit lue entièrement montre que l'on n'a pas d'égalité : par exemple, le langage des palindromes, ou 0^n sur un aphabet à deux lettres, ou 0^{2k} sur un alphabet à une lettre.

Exercise 5 Sur l'existence de fonction *dans un sens*

Une fonction *dans un sens* est une bijection f d'entiers à k-bit vers des entiers à k-bits telle que f est calculable en temps polynomial, mais pas f^{-1}. Prouver que pour toute fonction *dans un sens* f, on a

$$A \stackrel{\text{def}}{=} \{(x, y) \mid f^{-1}(x) < y\} \in (\mathsf{NP} \cap \mathsf{coNP}) \backslash \mathsf{P}\,.$$

Solution :

1. $A \in \mathsf{NP}$: considérons une machine de Turing machine qui, sur une entrée $w = (x, y)$, devine un nombre c (avec $|c| = |x|$) et vérifie en temps polynomial que $f(c) = x$ et $c < y$. Cette machine de Turing non déterministe s'exécute en temps polynomial et accepte le langage A.
2. $A \in \mathsf{coNP} \Leftrightarrow \{(x, y) \mid f^{-1}(x) \geq y\} \in \mathsf{NP}$, que l'on traite comme précédemment.
3. Supposons que $A \in \mathsf{P}$. On construit alors une machine de Turing s'exécutant en temps polynomial qui calcule f^{-1} : sur une entrée x telle que $|x| = n$, il y a 2^n possibilité pour la valeur de $f^{-1}(x)$. On considère une machine de Turing qui procède par dichotomie sur toutes les valeurs possibles v de $f^{-1}(x)$ jusqu'à ce qu'un v tel que $(x, v-1) \in A$ et $(x, v) \notin A$ soit trouvé, auquel cas on peut déduire que $f^{-1}(x) = v$. Comme on a au plus n tests nécessaires et que chaque test s'effectue en temps polynomial, cette machine de Turing s'exécute en temps polynomial.

Exercise 6 Argument de "bourrage" (*padding* en anglais)

1. Montrer que si $\mathsf{DSPACE}(n^c) \subseteq \mathsf{NP}$ pour un $c > 0$, alors $\mathsf{PSPACE} \subseteq \mathsf{NP}$.

   Indice : pour $L \in \mathsf{DSPACE}(n^k)$ on pourra considérer le langage $\tilde{L} = \{(x, 1^{|x|^{k/c}}) \mid x \in L\}$.
2. Déduisez-en que $\mathsf{DSPACE}(n^c) \neq \mathsf{NP}$. (On pourra utiliser le théorème de hiérarchie en espace de la feuille d'exercice précédente).

Solution :

1. Supposons que $\mathsf{DSPACE}(n^c) \subseteq \mathsf{NP}$ et considérons un langage $L \in \mathsf{PSPACE}$: on doit prouver que $L \in \mathsf{NP}$. Pour un k donné, on a $L \in \mathsf{DSPACE}(n^k)$. Soit M la machine de Turing décidant L en espace n^k. Considérons à présent le langage $\tilde{L} = \{(x, 1^{|x|^{k/c}}) \mid x \in L\}$ et la machine de Turing $\tilde{M}$ qui, sur une entrée w, vérifie qu'il est bien de la forme $w = (x, 1^\ell)$, vérifie que $\ell = |x|^{k/c}$, et si oui lance une simulation de M sur x. On peut remarquer que calculer $|x|^{k/c}$ utilise seulement k/c boucles imbriquées de 1 à $|x|$, ce qui peut s'effectuer en espace logarithmique vu que k/c est une "constante" qui dépend de M, pas de x. Alors, $\tilde{M}$ accepte $\tilde{L}$ et l'espace utilisé par $\tilde{M}$ est : $|x|^k = |1^{|x|^{k/c}}|^c \leq |w|^c$. De ce fait, $\tilde{L} \in \mathsf{DSPACE}(n^c) \subseteq \mathsf{NP}$. Donc $\tilde{L} \in \mathsf{NP}$. Comme on peut réduire L à $\tilde{L}$ en transformant x en $(x, 1^{|x|^{k/c}})$ en espace logarithmique, on a bien que $L \in \mathsf{NP}$.
2. Supposons que $\mathsf{DSPACE}(n^c) = \mathsf{NP}$, alors $\mathsf{DSPACE}(n^{c+1}) \subseteq \mathsf{PSPACE} = \mathsf{NP} = \mathsf{DSPACE}(n^c)$ ce qui est une contradiction avec le théorème de hiérarchie en espace (voir TD précédent).