Corrigé TD 1
Complexité

Luc Lapointe
luc.lapointe@ens-paris-saclay.fr
home.lmf.cnrs.fr/LucLapointe/

On rappelle quelques définitions de classes de complexité usuelles :
- $\mathrm{L} = \mathrm{SPACE}(\log n)$ est la classe des langages décidés par une machine de Turing déterministe utilisant un espace logarithmique.
- $\mathrm{P} = \cup_{k\in\mathbb{N}} \mathrm{TIME}(n^k) = \mathrm{TIME}(n^{O(1)})$ est la classe des langages décidés par une machine de Turing déterministe en temps polynomial.

On rappelle que l'espace utilisé par une machine de Turing est celui utilisé par sa bande de travail.

Exercice 1 - Comparaison de deux entiers

Trouver des algorithmes les plus simples possibles, en terme de clarté, parmi les algorithmes en temps polynomial qui répondent aux questions suivantes. Préciser leur complexité en temps et en espace.

1. **Entrée :** deux entiers n et k représentés en base 2 séparés par $\#$.
   Question : est-ce que $n = k$?
2. **Entrée :** deux entiers n et k représentés en base 2 séparés par $\#$.
   Question : est-ce que $n \leq k$?

Idée de solution

1. Écrire n sur une bande. Écrire k sur une deuxième bande. Lire les deux bandes simultanément et vérifier qu'elles sont identiques.
2. Écrire n sur une bande. Écrire k sur une autre. Lire les deux bandes en partant du bit de poids fort et s'arrêter dès que le bit de n est 1 et celui de k est 0, ou à la fin.

Exercice 2 - Addition

Trouver un algorithme en espace logarithmique répondant à la question suivante :

Entrée : deux entiers n et k représentés en base 2 séparés par $\#$.
Sortie : $n + k$?

Idée de solution

Impossible d'écrire n et k, ce serait en espace linéaire. On peut par contre faire de nombreux allers-retours entre n et k.

On fait l'addition comme l'addition posée apprise en primaire. Il faut un booléen qui retient s'il y a une retenue, et un compteur qui retient l'indice du bit en train d'être évalué.

Exercice 3 - Comparaison en espace logarithmique

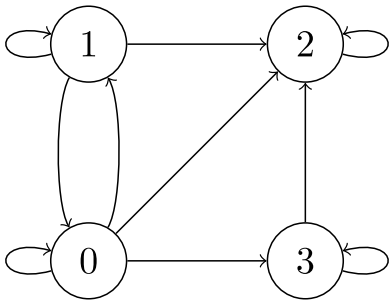
Trouver des algorithmes en espace logarithmique pour les problèmes de l'exercice 1.

Idée de solution

Impossible d'écrire n et k, ce serait en espace linéaire. On peut par contre les remplacer par un compteur i qui indique le numéro du bit en train d'être comparé, et faire de nombreux allers-retours entre le i^e bit de n et le i^e bit de k.

Exercice 4 - Représentation des graphes

Un graphe orienté est un couple $G = (S, A)$ où $S = \{0, ..., n-1\}$ est un ensemble fini de n sommets et $A \subseteq S \times S$ un ensemble d'arêtes. Il existe deux façons standards de représenter un graphe avec un alphabet fini Σ contenant au moins $\{0, 1\}$: les *listes d'adjacence* et les *matrices d'adjacence*.



1. Décrire ce que sont ces représentations.
2. Proposer un alphabet Σ utilisable pour ces deux représentations de graphes.
3. Décrire le graphe ci-dessus en liste et en matrice d'adjacence avec cet alphabet.
4. Montrer qu'il existe une fonction logspace (i.e. qui peut être implémentée par une MT qui utilise un espace logarithmique) qui à toute représentation d'un graphe G en liste d'adjacence associe une représentation de G en matrice d'adjacence.
   Réciproquement, montrer qu'il existe une fonction logspace qui à toute représentation d'un graphe G en matrice d'adjacence associe une représentation de G en liste d'adjacence.

Idée de solution

1. Une représentation en *liste d'adjacence* est une liste dont le i^e élément est la liste des voisins de i. Une représentation en *matrice d'adjacence* est une matrice dont la case (i, j) contient 1 ssi il y a une arête de i à j.
2. Je propose l'alphabet $\Sigma = \{[,], ,, 0, 1\}$.
3. Dans les représentations ci-dessous, 2 et 3 sont du sucre syntaxique pour décrire 10 et 11.
   Liste d'adjacence :

$$[[0, 1, 2, 3], [0, 1, 2], [2], [2, 3]]$$

Matrice d'adjacence :

$$[[1,1,1,1],[1,1,1,0],[0,0,1,0],[0,0,1,1]]$$

4. Il est possible de passer d'une liste d'adjacence à une matrice en retenant seulement la prochaine coordonnée de la matrice à remplir, donc en utilisant deux compteurs.
   Pour le passage de matrice à liste, il faut retenir les coordonnées de la matrice en train d'être lue, pour savoir quel nombre binaire écrire quand un 1 est croisé dans la matrice.

Exercice 5 - Temps polynomial exact

On appelle *monôme* une fonction de la forme $x \mapsto \lambda x^k$ où $\lambda, k \in \mathbb{N}$. On rappelle qu'une fonction f est dite *constructible* s'il existe une machine M qui produit, pour toute entrée de longueur n, la représentation unaire de $f(n)$ en temps $O(n + f(n))$ et en espace $O(f(n))$.

1. Montrer que tout monôme est constructible.
2. Montrer que pour f constructible et telle que $f(n) \geq n$, il existe $L \in \mathbb{N}$ tel que f est constructible en temps *exactement* $Lf(n)$.
3. Définissons Poly' comme la classe des langages L tels qu'il existe une MT M et un polynôme p avec M décidant L en temps exactement p (au lieu de au plus p). Montrer que Poly' $=$ P.

Idée de solution

1. On admet que $x \times y$ se calcule en $O(xy)$ et $x + y$ en $O(x + y)$. Pour arriver au résultat souhaité, on peut utiliser l'*exponentiation rapide* : pour calculer x^n, on calcule x^p pour p toutes les puissances de 2, jusqu'à la plus grande inférieure à n. Puis on les multiplie entre elles pour atteindre x^n, et on termine en multipliant par λ.

   Savoir jusqu'à quelle puissance multiplier et quelles puissances sommer est hard-codé, et n'a pas besoin d'être calculé à la volée.

   Ou alors, une deuxième solution, plus simple : calculer x, puis x^2, puis x^3... qui aboutit aussi à une complexité suffisamment petite.

2. Soit f constructible. Il existe donc une constante K et une machine de Turing qui la construit en temps borné par $K(n + f(n))$ et en espace $O(f(n))$. En particulier, cette machine la construit en temps borné par $2Kf(n)$.
   À partir de cette machine, on construit une machine M', avec deux rubans de plus. Cette machine M' imite M, et rajoute un 1 sur un ruban excédentaire à chaque étape de calcul sauf les n premières, et se comporte avec le deuxième ruban excédentaire comme avec le ruban de résultat. Une fois que M a terminé son calcul, il y a au plus $2Kf(n)$ caractères 1 sur le premier ruban excédentaire, et exactement $f(n)$ sur le second.

Cette machine M' peut ensuite effacer les caractères du premier ruban et du deuxième ruban en même temps, au rythme de un caractère sur le deuxième tous les $2K$ caractères sur le premier, tant qu'il y en a sur les deux. Puis prendre $4K$ étapes de calcul pour effacer chaque caractère restant sur la deuxième bande quand il n'en reste plus que sur la deuxième. Ceci donne une machine qui s'exécute en temps exactement $L := 4K$.

3. Poly' $\subseteq$ P est direct. Soit $L \in \mathrm{P}$, et p à coefficients *positifs* tel que M décide L en temps au plus p. On utilise la question précédente pour décider en temps exactement $4p$.

Exercice 6 - Temps d'arrêt de L

1. Montrer que toute MT qui calcule en espace logarithmique s'arrête en temps polynomial.
2. Définissons L$'$ comme la classe des langages semi-décidés en espace logarithmique (i.e. la machine peut ne pas s'arrêter sur x si x n'est pas dans le langage). Montrer que $\mathrm{L}' = \mathrm{L}$.

Idée de solution

1. Soit M une machine sur un alphabet Σ avec des états Q qui s'exécute en utilisant un espace au plus $\log(n)$. Une *configuration* est un triplet (w, q, w') où $w, w' \in \Sigma$ et $q \in Q$ qui décrit l'état d'une bande. Le nombre de configurations différentes possibles sur une entrée de taille n est

$$\sum_{i=0}^{\log(n)} |\Sigma|^i \ |Q| \ |\Sigma|^{\log(n)-i} = \log(n) \ |Q| \ |\Sigma|^{\log(n)}$$

   qui est polynomial en n. Or une machine déterministe qui termine ne peut passer deux fois par la même configuration (sinon ce serait un cycle infini). Quel que soit son nombre de bandes, M termine donc en temps au plus polynomial.
2. **Première idée** : Compter jusqu'à la borne ci-dessus. Il est bien possible de calculer ce polynôme en espace logarithmique.

   Deuxième idée : réexploiter la présence d'un cycle dans les exécutions longues en espace logarithmique. Pour se maintenir en espace logarithmique, une méthode de « lièvre et de la tortue » fonctionne. Soit M une machine sur un alphabet Σ avec des états Q qui semi-décide un problème en espace logarithmique. On peut construire une machine M' à partir de M mais avec son nombre de bandes de travail doublé. M' va imiter M une fois sur deux sur les premières bandes, et imiter M sur les deuxièmes. Toutes les deux étapes (donc toutes les transitions sur la première série de bandes qui n'avancent qu'une fois sur deux), on vérifie si les deux séries de bande sont dans la même configuration ou non. Ce sera forcément le cas à un moment si un cycle est présent, auquel cas on peut arrêter l'exécution de la machine et refuser.

Exercice 7 - Composition de fonctions logspace

Soient $h_1 : L_1 \mapsto L_2$ et $h_2 : L_2 \mapsto L_3$ deux fonctions calculables en espace logarithmique par des machines déterministes.

1. Donner la machine naïve calculant $h_2 \circ h_1$. Quel est l'espace utilisé par cette machine ?
2. Montrer que la fonction $h_2 \circ h_1 : L_1 \mapsto L_3$ peut aussi être calculée en espace logarithmique par une machine déterministe.
3. Rappeler un intérêt majeur de cette propriété.

Idée de solution

1. Calculer $h_1(x)$ avec la machine qui calcule h_1, puis $h_2(h_1(x))$ avec la machine qui calcule h_2 qui prend en entrée la sortie de la première. Mais la sortie de la première est d'une taille polynomiale plutôt que linéaire, donc le tout n'est plus en espace borné par log mais plutôt borné par $k \log$, où k est le degré du polynôme bornant la taille de la sortie.
2. Commencer par h_2, mais à chaque fois que l'information de la sortie de h_1 est nécessaire, elle est recalculée. Ainsi, seulement une cellule de la sortie de h_1 est retenue à tout instant.
3. Composer des réductions logspace donne encore une réduction logspace.