Fondements de l'informatique
Logique, modèles, et calculs

Chapitre: Complexité en espace mémoire

Cours CSC_41012_EP

de l'Ecole Polytechnique

Olivier Bournez

bournez@lix.polytechnique.fr

Version du 16 juillet 2024

Complexité en espace mémoire

Dans ce chapitre, on s'intéresse à une autre ressource critique dans les algorithmes : la mémoire. En fait, en théorie de la complexité, on parle plutôt *d'espace*, ou *d'espace mémoire*, pour désigner la mémoire.

On va commencer par voir comment on mesure la mémoire utilisée par un algorithme. On introduira alors les principales classes considérées en théorie de la complexité.

1 Espace polynomial

Dans toute cette section, nous énonçons un ensemble de définitions et de théorèmes sans preuve. Les preuves sont données dans la section suivante.

Introduisons l'analogue de TIME(t(n)) pour la mémoire :

Définition 1 (SPACE(t(n))**)** *Soit $t : \mathbb{N} \to \mathbb{N}$ une fonction. On définit la classe* SPACE(t(n)) *comme la classe des problèmes (langages) qui sont décidés par une machine de Turing en utilisant $\mathcal{O}(t(n))$ cases du ruban, où n est la taille de l'entrée.*

1.1 Classe PSPACE

On considère alors la classe des problèmes décidés en utilisant un espace mémoire polynomial.

Définition 2 PSPACE *est la classe des problèmes (langages) décidés en espace polynomial. En d'autres termes,*

$$\text{PSPACE} = \bigcup_{k \in \mathbb{N}} \text{SPACE}(n^k).$$

Remarque 1 *Comme dans le chapitre 12, on pourrait observer que cette notion ne dépend pas réellement du modèle de calcul utilisé, et que l'utilisation des machines de Turing comme modèle de base est relativement arbitraire.*

On peut aussi introduire l'analogue non déterministe :

Définition 3 (NSPACE(t(n))) *Soit $t : \mathbb{N} \to \mathbb{N}$ une fonction. On définit la classe* NSPACE(t(n))
comme celle des problèmes (langages) qui sont acceptés par une machine de Turing non déterministe en utilisant $\mathcal{O}(t(n))$ cases du ruban sur chacune des branches du calcul, où n est la taille de l'entrée.

Il serait alors naturel de définir

$$\text{NPSPACE} = \bigcup_{k \in \mathbb{N}} \text{NSPACE}(n^k),$$

mais il s'avère que la classe NPSPACE n'est autre que PSPACE.

Théorème 1 (Théorème de Savitch) NPSPACE = PSPACE.

La preuve de ce résultat se trouve dans la section 10.

1.2 Problèmes PSPACE-complets

La classe PSPACE possède des problèmes complets : le problème QBF (aussi appelé QSAT) consiste, étant donnée une formule du calcul propositionnel en forme normale conjonctive ϕ avec les variables $x_1, x_2, \cdots, x_n$ (c'est-à-dire une instance de SAT), à déterminer si

$$\exists x_1 \forall x_2 \exists x_3 \cdots \phi(x_1, \cdots, x_n) ?$$

Théorème 2 *Le problème* QBF *est* PSPACE*-complet.*

Nous ne prouverons pas ce résultat dans ce document.

Les jeux stratégiques sur les graphes donnent naturellement naissance à des problèmes PSPACE-complet.

Par exemple, le jeu GEOGRAPHY consiste à se donner un graphe orienté (fini) $G = (V, E)$. Le joueur 1 choisit un sommet u_1 du graphe. Le joueur 2 doit alors choisir un sommet v_1 tel qu'il y ait un arc de u_1 vers v_1. C'est alors au joueur 1 de choisir un autre sommet u_2 tel qu'il y ait un arc de v_1 vers u_2, et ainsi de suite. On n'a pas le droit de repasser deux fois par le même sommet. Le premier joueur qui ne peut pas continuer le chemin $u_1 v_1 u_2 v_2 \cdots$ perd. Le problème GEOGRAPHY consiste à déterminer étant donné un graphe G et un sommet de départ pour le joueur 1, s'il existe une stratégie gagnante pour le joueur 1.

Théorème 3 *Le problème* GEOGRAPHY *est* PSPACE*-complet.*

2 Espace logarithmique

Il s'avère que la classe PSPACE est énorme et contient tout P et aussi tout NP : imposer un espace mémoire polynomial est donc souvent peu restrictif.

C'est pourquoi on cherche souvent plutôt à parler d'un espace logarithmique : mais cela introduit une difficulté et un problème dans les définitions : en effet, une machine de Turing utilise au moins les cases qui contiennent son entrée, et donc la définition 1 ne permet pas de parler de fonctions $t(n) < n$.

C'est pour cela que l'on modifie cette définition avec la convention suivante : lorsque l'on mesure l'espace mémoire utilisé, par convention, on ne compte pas les cases de l'entrée.

Pour le faire, proprement, il faut donc remplacer la définition 1 par la suivante :

Définition 4 (SPACE(t(n))**)** *Soit* $t : \mathbb{N} \to \mathbb{N}$ *une fonction. On définit la classe* SPACE(t(n)) *comme la classe des problèmes (langages) qui sont décidés par une machine de Turing qui utilise deux rubans :*

- *le premier ruban contient l'entrée et est accessible en lecture seulement : il peut être lu, mais il ne peut pas être écrit ;*
- *le second est lui initialement vide et est accessible en lecture et écriture ;*

en utilisant $\mathcal{O}(t(n))$ *cases du second ruban, où n est la taille de l'entrée.*

On définit NSPACE(t(n)) *avec la même convention.*

Remarque 2 *Cette nouvelle définition ne change rien aux classes précédentes. Elle permet simplement de donner un sens aux suivantes.*

Définition 5 (LOGSPACE**)** *La classe* LOGSPACE *est la classe des langages et des problèmes décidés par une machine de Turing en espace logarithmique. En d'autres termes,*

$$\text{LOGSPACE} = \text{SPACE}(\log(n)).$$

Définition 6 (NLOGSPACE**)** *La classe* NLOGSPACE *est la classe des langages et des problèmes décidés par une machine de Turing en espace non déterministe logarithmique. En d'autres termes,*

$$\text{NLOGSPACE} = \text{NSPACE}(\log(n)).$$

Il s'avère que l'on a :

Théorème 4 $\text{LOGSPACE} \subset \text{NLOGSPACE} \subset \text{P} \subset \text{NP} \subset \text{PSPACE}$.

On sait par ailleurs que $\text{NLOGSPACE} \subsetneq \text{PSPACE}$ mais on ne sait pas lesquelles des inclusions intermédiaires sont strictes.

3 Quelques résultats et démonstrations

Cette section est consacrée à prouver un certain nombre des principaux résultats de la théorie de la complexité, en particulier sur les liens entre temps et mémoire. Observons que le théorème 4 découle des résultats qui suivent.

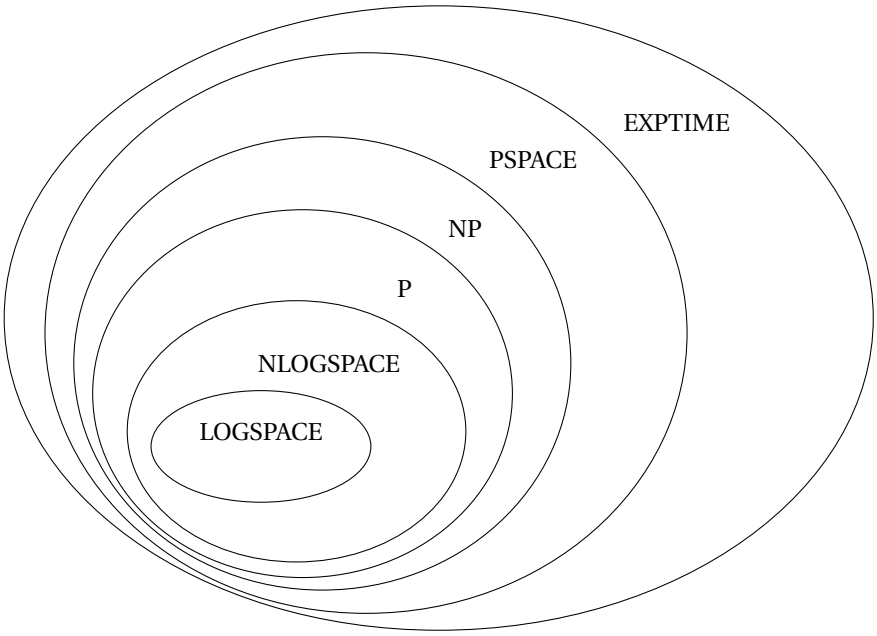

FIGURE 1 – Inclusions entre les classes de complexité

3.1 Préliminaires

Pour éviter de compliquer inutilement certaines preuves, nous nous limiterons à des fonctions $f(n)$ *de complexité propre* : on suppose que la fonction $f(n)$ est non décroissante, c'est-à-dire que $f(n+1) \geq f(n)$, et qu'il existe une machine de Turing qui prend en entrée w et qui produit en sortie $\mathbf{1}^{f(n)}$ en temps $\mathcal{O}\left(n+f(n)\right)$ et en espace $\mathcal{O}\left(f(n)\right)$, où $n = \text{length}\, w$.

Remarque 3 *Cela n'est pas vraiment restrictif, car toutes les fonctions usuelles non décroissantes, comme* $\log(n)$, n, n^2, $\cdots$, $n\log n$, $n!$ *vérifient ces propriétés. En outre, ces fonctions sont stables par somme, produit, et exponentielle.*

Remarque 4 *Nous avons besoin de cette hypothèse, car la fonction $f(n)$ pourrait ne pas être calculable, et donc il pourrait par exemple ne pas être possible d'écrire un mot de longueur $f(n)$ dans un des algorithmes qui suivent.*

Remarque 5 *Dans la plupart des assertions qui suivent, on peut se passer de cette hypothèse, au prix de quelques complications dans les preuves.*

3.2 Relations triviales

Un problème déterministe étant un problème non déterministe particulier, on a :

Théorème 5 SPACE(f(n)) ⊂ NSPACE(f(n)).

D'autre part :

Théorème 6 TIME(f(n)) ⊂ SPACE(f(n)).

Démonstration: Une machine de Turing écrit au plus une case à chaque étape. L'espace mémoire utilisé reste donc linéaire en le temps utilisé. Rappelons que l'on ne compte pas l'entrée dans l'espace mémoire. □

3.3 Temps non déterministe vs déterministe

De façon plus intéressante :

Théorème 7 *Pour tout langage de* NTIME(f(n)), *il existe un entier c tel que ce langage soit dans* TIME($c^{f(n)}$). *Si l'on préfère :*

$$\text{NTIME(f(n))} \subset \bigcup_{c\in\mathbb{N}} \text{TIME}(c^{f(n)}).$$

Démonstration (principe): Soit L un problème de NTIME(f(n)). En utilisant le principe que l'on a utilisé dans le chapitre précédent, on sait qu'il existe un problème A tel que pour déterminer si un mot w de longueur n est dans L, il suffit de savoir s'il existe un mot $u \in \Sigma^*$ avec $\langle w,u\rangle \in A$, ce dernier test pouvant se faire en temps $f(n)$, où $n = \text{length}\, w$. Puisqu'en temps $f(n)$ on ne peut pas lire plus que $f(n)$ lettres de u, on peut se limiter aux mots u de longueur $f(n)$. Tester si $\langle w,u\rangle \in A$ pour tous les mots $u \in \Sigma^*$ de longueur $f(n)$ se fait facilement en temps $\mathcal{O}\left(c^{f(n)}\right) * \mathcal{O}\left(f(n)\right) = \mathcal{O}\left(c^{f(n)}\right)$, où $c > 1$ est le cardinal de l'alphabet Σ de la machine : tester tous les mots u peut par exemple se faire en comptant en base c. □

Remarque 6 *Pour écrire le premier u à tester de longueur $f(n)$, nous utilisons le fait que cela doit être possible : c'est le cas, si l'on suppose $f(n)$ de complexité propre. On voit donc l'intérêt ici de cette hypothèse implicite. Nous éviterons de discuter ce type de problèmes dans la suite, qui ne se posent pas de toute façon pour les fonctions $f(n)$ usuelles.*

3.4 Temps non déterministe vs espace

Théorème 8 NTIME(f(n)) ⊂ SPACE(f(n)).

Démonstration: On utilise exactement le même principe que dans la preuve précédente, si ce n'est que l'on parle d'espace. Soit L un problème de NTIME(f(n)). En utilisant la même idée que dans le chapitre précédent, on sait qu'il existe un problème A tel que pour déterminer si un mot w de longueur n est dans L, il suffit de savoir s'il existe $u \in \Sigma^*$ de longueur $f(n)$ avec $\langle w,u\rangle \in A$: on utilise un espace $\mathcal{O}\left(f(n)\right)$

pour générer un à un les mots $u \in \Sigma^*$ de longueur $f(n)$ (par exemple en comptant en base c) puis on teste pour chacun si $\langle w, u\rangle \in A$, ce dernier test se faisant en temps $f(n)$, donc espace $f(n)$. Le même espace pouvant être utilisé pour chacun des mots u, au total cela se fait au total en espace $\mathcal{O}\left(f(n)\right)$ pour générer les u plus $\mathcal{O}\left(f(n)\right)$ pour les tests, soit $\mathcal{O}\left(f(n)\right)$. □

3.5 Espace non déterministe vs temps

Le problème de décision REACH suivant jouera un rôle important : on se donne un graphe orienté $G = (V, E)$, deux sommets u et v, et on cherche à décider s'il existe un chemin entre u et v dans G. On peut facilement se persuader que REACH est dans P.

A toute machine de Turing M (déterministe ou non) est associé un graphe orienté, son graphe des configurations, où les sommets correspondent aux configurations et les arcs correspondent à la fonction d'évolution en un pas de M, c'est-à-dire à la relation $\vdash$ entre configurations.

Chaque configuration X peut se décrire par un mot $[X]$ sur l'alphabet de la machine : si on fixe l'entrée w de longueur n, pour un calcul en espace $f(n)$, il y a moins de $\mathcal{O}\left(c^{f(n)}\right)$ sommets dans ce graphe G_w, où $c > 1$ est le cardinal de l'alphabet de la machine.

Un mot w est accepté par la machine M si et seulement s'il y a un chemin dans ce graphe G_w entre l'état initial $X[w]$ codant l'entrée w, et un état acceptant. On peut supposer sans perte de généralité qu'il y a une unique configuration acceptante X^*. Décider l'appartenance d'un mot w au langage reconnu par M est donc résoudre le problème REACH sur $\langle G_w, X[w], X^*\rangle$.

On va alors traduire sous différentes formes tout ce que l'on sait sur le problème REACH. Tout d'abord, il est clair que le problème REACH se résout par exemple en temps et espace $\mathcal{O}\left(n^2\right)$, où n est le nombre de sommets, par un parcours en profondeur.

On en déduit :

Théorème 9 *Si $f(n) \geq \log n$, alors*

$$\mathrm{NSPACE(f(n))} \subset \bigcup_{c\in\mathbb{N}} \mathrm{TIME}(\mathrm{c}^{\mathrm{f(n)}}).$$

Démonstration: Soit L un problème de NSPACE(f(n)) reconnu par la machine de Turing non déterministe M. Par la discussion plus haut, on peut déterminer si $w \in L$ en résolvant le problème REACH sur $\langle G_w, X[w], X^*\rangle$: on a dit que cela pouvait se fait en temps quadratique en le nombre de sommets, soit en temps $\mathcal{O}\left(c^{2\mathcal{O}(f(n))}\right)$, où $c > 1$ est le cardinal de l'alphabet de la machine.

□

3.6 Espace non déterministe vs espace déterministe

On va maintenant affirmer que REACH se résout en espace $\log^2(n)$.

Proposition 1 REACH ∈ SPACE($\log^2$n).

Démonstration: Soit $G = (V, E)$ le graphe orienté en entrée. Étant donnés deux sommets x et y de ce graphe, et i un entier, on note $CHEMIN(x, y, i)$ si et seulement s'il y a un chemin de longueur inférieure à 2^i entre x et y. On a $\langle G, u, v\rangle \in$ REACH si et seulement si on a $CHEMIN(u, v, \log(n))$, où n est le nombre de sommets. Il suffit donc de savoir décider la relation $CHEMIN$ pour décider REACH.

L'astuce est de calculer $CHEMIN(x, y, i)$ récursivement en observant que l'on a la relation $CHEMIN(x, y, i)$ si et seulement s'il existe un sommet intermédiaire z tel que $CHEMIN(x, z, i-1)$ et $CHEMIN(z, y, i-1)$. On teste alors à chaque niveau de la récursion chaque sommet possible z.

Pour représenter chaque sommet, il faut $\mathcal{O}\left(\log(n)\right)$ bits. Pour représenter x, y, i, il faut donc $\mathcal{O}\left(\log(n)\right)$ bits. Cela donne une récurrence de profondeur $\log(n)$, chaque étape de la récurrence nécessitant uniquement de stocker un triplet x, y, i et de tester chaque z de longueur $\mathcal{O}\left(\log(n)\right)$. Au total, on utilise donc un espace $\mathcal{O}\left(\log(n)\right) * \mathcal{O}\left(\log(n)\right) = \mathcal{O}\left(\log^2(n)\right)$. □

Théorème 10 (Savitch) *Si $f(n) \geq \log(n)$, alors*

$$\text{NSPACE(f(n))} \subset \text{SPACE(f(n)}^2\text{)}.$$

Démonstration: On utilise cette fois l'algorithme précédent pour déterminer s'il y a un chemin dans le graphe G_w entre $X[w]$ et X^*.

On remarquera que l'on a pas besoin de construire explicitement le graphe G_w mais que l'on peut utiliser l'algorithme précédent à la volée : plutôt que d'écrire complètement le graphe G_w, et ensuite de travailler en lisant dans cette écriture du graphe à chaque fois s'il y a un arc entre un sommet X et un sommet X', on peut de façon paresseuse, déterminer à chaque fois que l'on fait un test si $C_w(X, X') = 1$. □

Corollaire 1 PSPACE = NPSPACE.

Démonstration: On a $\bigcup_{c\in\mathbb{N}} \text{SPACE(n}^c\text{)} \subset \bigcup_{c\in\mathbb{N}} \text{NSPACE(n}^c\text{)}$, par le théorème 5, et

$$\bigcup_{c\in\mathbb{N}} \text{NSPACE(n}^c\text{)} \subset \bigcup_{c\in\mathbb{N}} \text{SPACE(n}^{2c}\text{)} \subset \bigcup_{c\in\mathbb{N}} \text{SPACE(n}^c\text{)}$$

par le théorème précédent. □

3.7 Espace logarithmique non déterministe

En fait, on peut encore dire plus sur le problème REACH.

Théorème 11 REACH ∈ NLOGSPACE.

Démonstration (principe): Pour déterminer s'il y a un chemin entre u et v dans un graphe G, on devine de façon non déterministe le chemin sommet par sommet. Cela nécessite uniquement de garder le sommet que l'on est en train de visiter en plus de u et de v. Chaque sommet se codant par $\mathcal{O}\left(\log(n)\right)$ bits, l'algorithme est en espace $\mathcal{O}\left(\log(n)\right)$. □

Définition 7 *Soit $\mathscr{C}$ une classe de complexité. On note co-$\mathscr{C}$ pour la classe des langages dont le complémentaire est dans la classe $\mathscr{C}$.*

On parle ainsi de problème coNP, coNLOGSPACE, etc...
En fait, on peut montrer :

Théorème 12 *Le problème* REACH *est dans* coNLOGSPACE.

On en déduit :

Théorème 13 NLOGSPACE = coNLOGSPACE.

Démonstration: Il suffit de montrer que coNLOGSPACE ⊂ NLOGSPACE. L'inclusion inverse en découle car un langage L de NLOGSPACE aura son complémentaire dans la classe coNLOGSPACE et donc aussi dans NLOGSPACE, donc L sera dans coNLOGSPACE.

Maintenant, pour décider si un mot w doit être accepté par un langage de la classe coNLOGSPACE, on peut utiliser la machine non déterministe du théorème précédent qui utilise un espace logarithmique pour déterminer s'il existe un chemin entre $X[w]$ et X^* dans le graphe G_w. □

En fait, selon le même principe on peut montrer plus généralement.

Théorème 14 *Soit $f(n)$ une fonction telle que $f(n) \geq \log(n)$. Alors*

$$\text{NSPACE(f(n))} = \text{coNSPACE(f(n))}\,.$$

4 Résultats de séparation

4.1 Théorèmes de hiérarchie

On dit qu'une fonction $f(n) \geq \log(n)$ est *constructible en espace*, si la fonction qui envoie $\mathbf{1}^n$ sur la représentation binaire de $\mathbf{1}^{f(n)}$ est calculable en espace $\mathcal{O}\left(f(n)\right)$.

La plupart des fonctions usuelles sont constructibles en espace. Par exemple, n^2 est constructible en espace puisqu'une machine de Turing peut obtenir n en binaire en comptant le nombre de $\mathbf{1}$, et écrire n^2 en binaire par n'importe quelle méthode pour multiplier n par lui-même. L'espace utilisé est certainement en $\mathcal{O}\left(n^2\right)$.

Théorème 15 (Théorème de hiérarchie en espace) *Pour toute fonction $f : \mathbb{N} \to \mathbb{N}$ constructible en espace, il existe un langage L qui est décidable en espace $\mathcal{O}\left(f(n)\right)$ mais qui ne l'est pas en espace $o(f(n))$.*

Démonstration: On considère le langage (très artificiel) L qui est décidé par la machine B suivante :

- sur une entrée w de taille n, B calcule $f(n)$ en utilisant la constructibilité en espace de f, et réserve un espace $f(n)$ pour la simulation qui va venir ;
- si w n'est pas de la forme $\langle A\rangle \mathbf{10}^*$, où $\langle A\rangle$ est le codage d'une machine A, alors la machine B refuse ;
- sinon, B simule la machine A sur le mot w pendant au plus $2^{f(n)}$ étapes pour déterminer si A accepte en ce temps avec un espace inférieur à $f(n)$:
  - si A accepte en ce temps et cet espace, alors B refuse ;
  - sinon B accepte.

Par construction, L est dans SPACE(f(n)), car la simulation n'introduit qu'un facteur constant dans l'espace nécessaire : plus concrètement, si A utilise un espace $g(n) \leq f(n)$, alors B utilise un espace au plus $cg(n)$ pour une constante c.

Supposons que L soit décidé par une machine de Turing A en espace $g(n)$ avec $g(n) = o(f(n))$. Il doit exister un entier n_0 tel que pour $n \geq n_0$, on ait $dg(n) < f(n)$ pour une constante d. Par conséquent, la simulation par B de A sera bien complète sur une entrée de longueur n_0 ou plus.

Considérons ce qui se passe lorsque B est lancée sur l'entrée $\langle A\rangle \mathbf{10}^{n_0}$. Puisque cette entrée est de taille plus grande que n_0, B répond l'inverse de la machine A sur la même entrée. Donc B et A ne décident pas le même langage, et donc la machine A ne décide pas L, ce qui mène à une contradiction.

Par conséquent L n'est pas décidable en espace $o(f(n))$. □

Autrement dit :

Théorème 16 (Théorème de hiérarchie en espace) *Soient $f, f' : \mathbb{N} \to \mathbb{N}$ des fonctions constructibles en espace telles que $f(n) = o(f'(n))$.*
Alors l'inclusion SPACE(f) ⊂ SPACE(f') *est stricte.*

Sur le même principe, on peut prouver :

Théorème 17 (Théorème de hiérarchie en espace) *Soient $f, f' : \mathbb{N} \to \mathbb{N}$ des fonctions constructibles en espace telles que $f(n) = o(f'(n))$.*
Alors l'inclusion NSPACE(f) ⊂ NSPACE(f') *est stricte.*

4.2 Applications

On en déduit :

Théorème 18 NLOGSPACE $\subsetneq$ PSPACE.

Démonstration: La classe NLOGSPACE est complètement incluse dans la classe SPACE($\log^2$ n) par le théorème de Savitch. Hors cette dernière est un sous-ensemble strict de SPACE(n), qui est inclus dans PSPACE. □

Sur le même principe, on obtient :

Définition 8 *Soit*

$$\text{EXPSPACE} = \bigcup_{c\in\mathbb{N}} \text{SPACE}(2^{n^c}).$$

Théorème 19 PSPACE $\subsetneq$ EXPSPACE.

Démonstration: La classe PSPACE est complètement incluse dans, par exemple, la classe SPACE($n^{\log(n)}$). Hors cette dernière est un sous-ensemble strict de SPACE(2^n), qui est inclus dans EXPSPACE. □

5 Exercices

Exercice 1 *(corrigé page 248) Montrer que si tout langage* NP*-dur est* PSPACE*-dur, alors* PSPACE = NP.

6 Notes bibliographiques

Lectures conseillées Pour aller plus loin sur les notions évoquées dans ce chapitre, nous suggérons [Sipser, 1997], [Papadimitriou, 1994] ainsi que le livre en français [Lassaigne & de Rougemont, 2004].

Pour un ouvrage de référence contenant les derniers résultats du domaine, nous renvoyons à [Arora & Barak, 2009].

Bibliographie Ce chapitre contient des résultats standards en complexité. Nous nous sommes essentiellement inspirés de leur présentation dans [Sipser, 1997] et [Papadimitriou, 1994].

Index

Bibliographie

[Arora & Barak, 2009] Arora, S. & Barak, B. (2009). *Computational Complexity : A Modern Approach*. Cambridge University Press. https://doi.org/10.1017/cbo9780511804090

[Lassaigne & de Rougemont, 2004] Lassaigne, R. & de Rougemont, M. (2004). *Logic and complexity*. Discrete Mathematics and Theoretical Computer Science. Springer. https://doi.org/10.1007/978-0-85729-392-3

[Papadimitriou, 1994] Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley.

[Sipser, 1997] Sipser, M. (1997). *Introduction to the Theory of Computation*. PWS Publishing Company.