Complexité des algorithmes et notation grand O

Ce texte sur l'efficacité des algorithmes se divise en deux parties. Notre premier but est de pouvoir comparer l'efficacité de différents algorithmes. Comment y arriver ?

- Il faut représenter la complexité de chacun des algorithmes par une fonction. (Ce travail est fait à la section 2.3 du livre de Rosen et nous nous y attarderons pas ici.)
- La notation grand O indique en quelque sorte « l'ordre de grandeur » des fonctions. Cela permet ensuite de comparer rapidement les fonctions pour savoir laquelle correspond à un algorithme plus efficace.

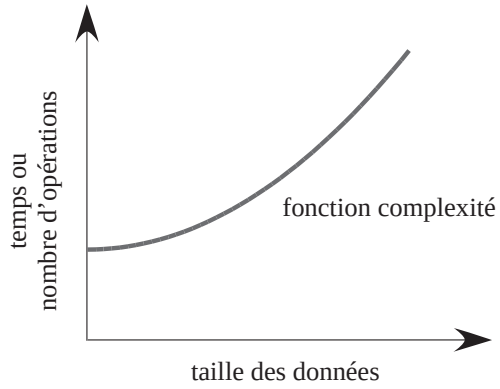
Dans un deuxième temps, nous nous poserons d'autres questions très importantes : *Est-ce que chaque problème peut être résolu par un algorithme ? Parmi les problèmes résolubles, quels sont ceux qui peuvent l'être par un algorithme efficace ?* Puis nous discuterons des problèmes NP-complets et de la question à un million de dollars à ce sujet.

Mesure de la complexité (en temps)

Le temps d'exécution d'un algorithme donné dépend principalement de

- la machine utilisée : langage, OS, compilateur...
- les données auxquelles l'algorithme est appliqué.

Nous allons utiliser une mesure plus abstraite, qui ne dépend pas de la machine ni des données elles-mêmes, mais plutôt de la *taille* des données (taille de l'input). Par exemple, le temps d'exécution d'un algorithme de tri dépend de la longueur de la liste à trier. On utilisera donc une fonction pour décrire la complexité d'un algorithme.



Exemple 1

En analysant l'algorithme de fouille linéaire (voir exemple 2, section 2.3), on constate que l'algorithme nécessite *au plus*[1] $f(n) = 2n+2$ comparaisons pour traiter une liste de taille n, tandis que l'algorithme de fouille dichotomique (binary search) nécessite au plus $g(n) = 2\log_2(n)+2$ comparaisons. Quel algorithme sera plus efficace pour traiter des grosses listes ?

[1] La fouille linéaire sera très efficace pour obtenir le numéro de téléphone de Karl Aach dans le bottin mais beaucoup moins pour obtenir celui de Steven Zuk ! Nous choisirons ici d'étudier la complexité dans le pire des cas (worst case complexity).

Exercice 1

Si l'algorithme A nécessite $f(n)$ opérations pour résoudre un problème de taille n et l'algorithme B en nécessite $g(n)$, lequel sera plus efficace pour résoudre les gros problèmes ?

(a) $f(n) = n^3 + 3$ et $g(n) = 25 + n^2$

(b) $f(n) = 2^n + 4n^3$ et $g(n) = 10n^4$

Pour répondre à cette question, vous avez peut-être tracé un graphe, observé un tableau, ou peut-être avez-vous simplement utilisé vos connaissances antérieures sur les fonctions pour répondre immédiatement à la question. Nous allons maintenant nous doter d'outils nous permettant de comparer très rapidement deux fonctions pour savoir laquelle correspond à un algorithme plus efficace.

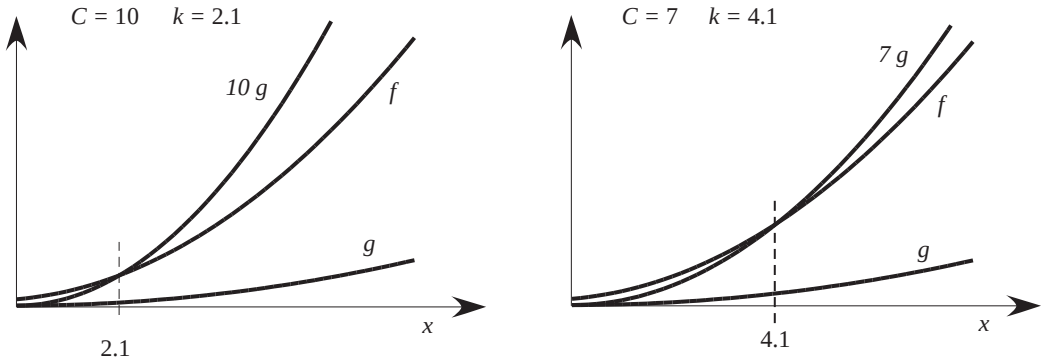
Notation grand O

Définition 1 p. 132

Soit f et g deux fonctions de R vers R ou de N vers R. On dit que $f(x)$ est $O(g(x))$, et on utilise parfois la notation $f(x) \in O(g(x))$, si f est éventuellement dépassée par un multiple de g, c'est-à-dire s'il existe des constantes C et k telles que

$$\forall\, x > k,\ |f(x)| \leq C|g(x)|$$

La borne k, appelée *seuil*, permet d'ignorer le comportement des fonctions pour les données de petites tailles (dans ces cas, la complexité de l'algorithme est souvent dominée par des opérations d'initialisation qui deviennent négligeables pour des données plus grandes).



La constante C, appelée *facteur*, permet de faire abstraction de vitesse de la machine utilisée.

Les nombres k et C sont appelés *témoins* de la relation $f(x) \in O(g(x))$.

Il suffit de trouver *une* paire de témoins pour prouver que $f(x)$ est $O(g(x))$. Mais il y a une infinité de choix possibles. Alors il sera très normal que vos réponses diffèrent de celles du livre ! Par exemple, pour prouver que $f(x) = 5x^2 + 6x + 9$ est $O(x^2)$, je peux fournir comme témoins $C = 7$ et $k = 4.1$ ou $C = 10$ et $k = 2.1$ comme l'indique la figure ci-dessus.

Le nombre d'opérations requises pour résoudre un problème ne peut être négatif. Ainsi, dans le cas particulier des fonctions de complexité des algorithmes, les valeurs absolues de la définition ne sont pas nécessaires car ces fonctions prennent toujours des valeurs positives.

Exemple 2

Montrer que $7x^2$ est $O(x^3)$ en fournissant les témoins. Voir page 135 de Rosen.

Il peut être laborieux de trouver les témoins k et C d'une relation grand O. Et dans la pratique, la valeur de ces témoins n'a pas d'importance. L'important est de savoir que $f(x)$ est $O(g(x))$. Voici quelques théorèmes permettant d'éviter la recherche de témoins.

Théorème 1 (N'apparaît pas dans Rosen)

Si

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = b \text{ et } b > 0$$

alors $f(x) \in O(g(x))$ et $g(x) \in O(f(x))$.

Si

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$$

alors $f(x) \in O(g(x))$ et $g(x) \notin O(f(x))$.

Exemple 3

Soit $f(x) = 7x^2$ et $g(x) = x^3$. Est-ce que $f(x) \in O(g(x))$? Est-ce que $g(x) \in O(f(x))$?

$$\begin{aligned} \lim_{x\to\infty} \frac{f(x)}{g(x)} &= \lim_{x\to\infty} \frac{7x^2}{x^3} \\ &= \lim_{x\to\infty} \frac{7}{x} \\ &= 0 \end{aligned}$$

donc

$$7x^2 \in O(x^3) \text{ mais } x^3 \notin O(7x^2)$$

Vous pouvez lire aux pages 135 et 136 une preuve équivalente utilisant uniquement la notion de témoins (et non celle de limite). Mais puisque vous avez tous eu la chance de suivre un cours de calcul différentiel, pourquoi ne pas en profiter pour gagner du temps ?

Exemple 4

Soit $f(n) = \log_2(n)$ et $g(n) = n$.

$$\begin{aligned} \lim_{n\to\infty} \frac{f(n)}{g(n)} &= \lim_{n\to\infty} \frac{\log_2(n)}{n} && \text{qui est de la forme } \frac{\infty}{\infty} \\ &= \lim_{n\to\infty} \frac{\frac{1}{n\ln(2)}}{1} && \text{par la règle de l'Hopital} \\ &= 0 \end{aligned}$$

donc

$$\log_2(n) \in O(n) \text{ mais } n \notin O(\log_2(n))$$

La même démarche fonctionne pour un logarithme de n'importe quelle base.

Exemple 5

Soit $f(n) = 2^n$ et $g(n) = 3^n$.

$$\begin{aligned}\lim_{n\to\infty} \frac{f(n)}{g(n)} &= \lim_{n\to\infty} \frac{2^n}{3^n} \\ &= \lim_{n\to\infty} \left(\frac{2}{3}\right)^n \\ &= 0\end{aligned}$$

donc

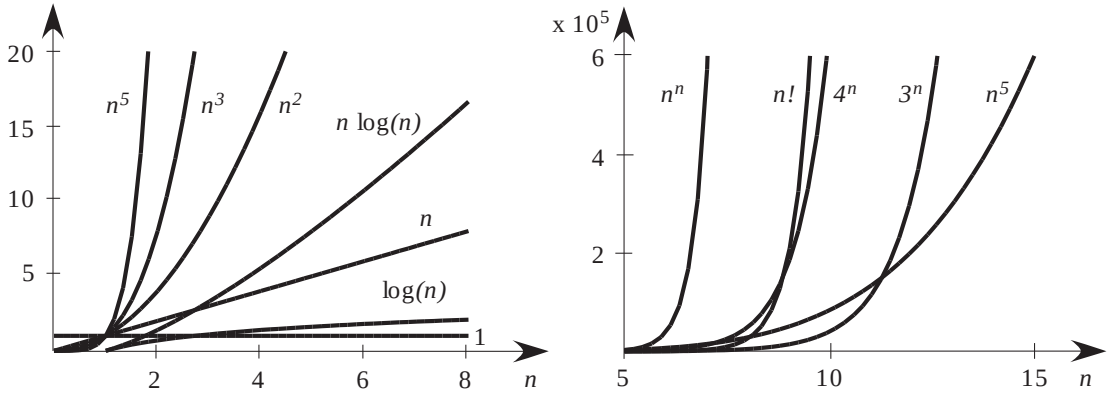
$$2^n \in O(3^n) \text{ mais } 3^n \notin O(2^n)$$

Théorème 2

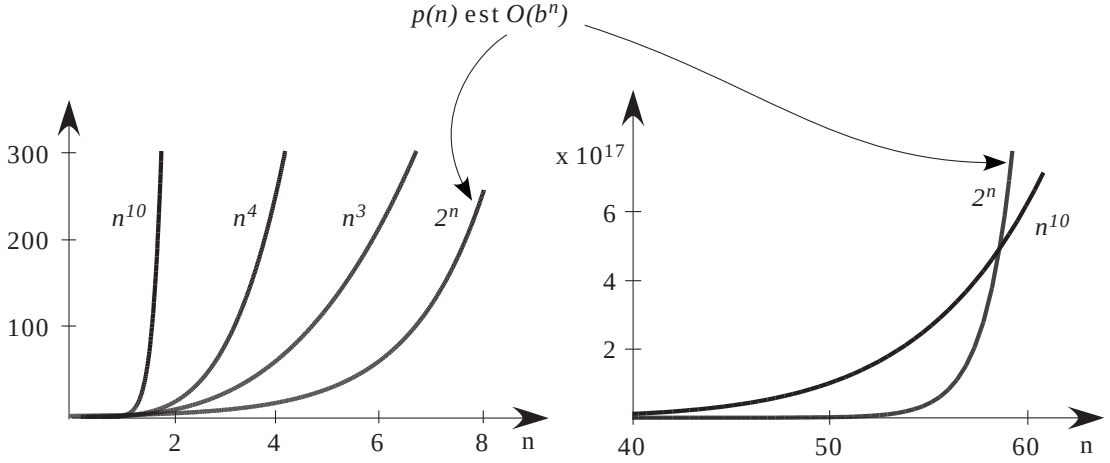
Dans la liste de fonctions ci-dessous, chaque fonction est grand O des fonctions qui sont situées plus à droite mais n'est pas grand O des fonctions situées plus à gauche.

$$1,\ \log(n),\ n,\ n\log(n),\ n^2,\ n^3,\ n^4,\ \ldots,\ 2^n,\ 3^n,\ 4^n,\ \ldots,\ n!,\ n^n$$

Les figures suivantes illustrent la croissance des fonctions de la liste.



Une exponentielle de base b supérieure à 1 arrive toujours à dépasser un polynôme $p(n)$, même si la base est très près de 1 et le degré du polynôme est très grand :



Théorème 3 (Grand O de la somme)

Si $f_1(x) \in O(g_1(x))$ et $f_2(x) \in O(g_2(x))$ et si $g_1(x) \in O(g_2(x))$ alors

$$(f_1 + f_2)(x) \in O(g_2(x))$$

Théorème 4 (Grand O du produit)

Si $f_1(x) \in O(g_1(x))$ et $f_2(x) \in O(g_2(x))$ alors

$$f_1 f_2(x) \in O(g_1 g_2(x))$$

Théorème 5 (Grand O d'un polynôme)

Si $p(x)$ est un polynôme de degré d alors

$$p(x) \in O(x^d)$$

Exemple 6

Soit

$$f(x) = \sqrt{7}x^6 + 7x^5 + \pi x^3 - 194x^2 - 2112$$

La fonction f est un polynôme de degré 6. Donc

$$\sqrt{7}x^6 + 7x^5 + \pi x^3 - 194x^2 - 2112 \in O(x^6)$$

Exemple 7

Soit

$$f(x) = x^2 + 5^x$$

Trouvez une fonction $g(x)$, la plus simple possible, telle que $f(x) \in O(g(x))$.

La fonction f est une somme. On peut traduire le théorème de la somme ainsi : *c'est la fonction dont le grand O domine les autres qui l'emporte.* Dans le cas de f, quelle fonction domine l'autre ? Allons voir la liste du théorème 2 : c'est l'exponentielle 5^x qui domine la puissance x^2, au sens où $x^2 \in O(5^x)$. Ainsi,

$$x^2 + 5^x \in O(5^x)$$

La fonction cherchée est donc $g(x) = 5^x$.

Exemple 8

Soit

$$f(x) = 4x^2 + 3x + 7 + 6 \cdot 3^x + 5\log(x)$$

Trouvez une fonction $g(x)$, la plus simple possible, telle que $f(x) \in O(g(x))$.

On sait que

$$4x^2 + 3x + 7 \in O(x^2)$$

car c'est un polynôme de degré 2 (théorème 5),

$$6 \cdot 3^x \in O(3^x)$$

car la limite du quotient est 6 (théorème 1), ou en utilisant les témoins $C = 6$ et $k = 0$,

$$5\log(x) \in O(\log(x))$$

car la limite du quotient est 5 (théorème 1), ou en utilisant les témoins $C = 5$ et $k = 0$.

D'après la liste du théorème 2, c'est 3^x qui domine. Ainsi, d'après le théorème de la somme,

$$4x^2 + 3x + 7 + 6 \cdot 3^x + 5\log(x) \in O(3^x)$$

La fonction cherchée est donc $g(x) = 3^x$.

Exemple 9

Soit

$$f(n) = (14n + 3)\log(n) + 3n^2$$

Trouvez une fonction $g(n)$, la plus simple possible, telle que $f(n) \in O(g(n))$.

On sait que

$$14n + 3 \in O(n)$$

car c'est un polynôme de degré 1. Donc, par le théorème du produit,

$$(14n + 3)\log(n) \in O(n\log(n))$$

On a $3n^2 \in O(n^2)$, $(14n + 3)\log(n) \in O(n\log(n))$ et, en regardant la liste du théorème 2 on voit que n^2 domine $n\log(n)$. Ainsi, d'après le théorème sur la somme,

$$(14n + 3)\log(n) + 3n^2 \in O(n^2)$$

La fonction cherchée est donc $g(n) = n^2$.

Calculabilité et complexité

Nous avons déjà discuté en classe du fait qu'il existe des problèmes qui ne peuvent être résolus par un programme. La preuve de cette affirmation consiste à montrer que l'ensemble des problèmes est non dénombrable alors que l'ensemble des programmes est dénombrable. Il est donc impossible d'établir une bijection entre ces deux ensembles : il y a plus de problèmes que de programmes possibles. Ceci dit, cette preuve ne fournit pas un problème concret non résoluble.

En 1936, Alan Turing a fourni un exemple concret en démontrant que le fameux problème de l'arrêt n'est pas résoluble par un programme.[2] Il serait donc totalement inutile de mettre sur pied une équipe d'ingénieurs en informatique pour tenter de résoudre un tel problème, comme il serait inutile mettre sur pied une équipe d'ingénieurs en mécanique pour construire une machine à mouvement perpétuel.

Quels sont les problèmes résolubles ? Non résolubles ? Voilà une des questions que traite la théorie de la calculabilité.

De plus, parmi les problèmes résolubles, ***quels sont les problèmes résolubles par un algorithme efficace ?*** Voilà une autre question importante, traitée par la théorie de la complexité. Mais qu'entend-on au juste par algorithme *efficace* ? Il est clair qu'un algorithme dont la complexité est exponentielle n'est *pas* efficace. Il est plus difficile de s'entendre sur le sens du mot *efficace* mais on convient généralement qu'il s'agit d'un algorithme dont la complexité est au plus polynomiale : $O(n^d)$ peu importe le degré. (Dans les faits, le degré du polynôme devra être petit pour que l'algorithme soit réellement efficace. Mais ceci n'est pas important ici.)

Il existe toute une gamme de problèmes importants pour lesquels aucun algorithme à complexité polynomiale n'a été trouvé. Et plusieurs de ces problèmes, quoique fort différents, sont équivalents au sens ou si l'un de ces problèmes est résoluble en temps polynomial, alors ils le sont tous. Ces problèmes sont qualifiés de **NP-complets**. En voici quelques-uns qui seront rencontrés en MAT210.

1. En théorie des graphes, le problème du circuit hamiltonien (HC) : existe-t-il un circuit fermé permettant de parcourir chaque sommet d'un graphe une et une seule fois ?

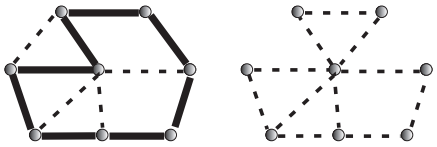


FIG. 1 – Le graphe de gauche contient un circuit hamiltonien.

2. En théorie des graphes, le problème du voyageur de commerce (travelling salesman, TS) : quel est le plus court chemin permettant de parcourir un ensemble de villes et de revenir à son point de départ ?

3. En logique, le problème de la satisfaisabilité d'une proposition (SAT) : étant donnée une proposition sous forme normale conjonctive, comme par exemple

$$(p \vee \neg q \vee s) \ \wedge \ (\neg p \vee q \vee s),$$

existe-t-il une fonction d'interprétation qui la rend vraie ? En d'autres mots, y a-t-il un choix de valeurs de vérité pour les propositions simples p, q et s qui rend la proposition composée vraie ?

Il existe un algorithme très simple qui permet de répondre à cette dernière question : vérifier si la table de vérité de la proposition composée contient une ligne Vrai. Mais comme nous l'avons vu au chapitre 1, si la proposition contient n propositions simples, la table de vérité comporte 2^n lignes. L'algorithme décrit n'est donc malheureusement pas polynomial.

Il en va de même des autres problèmes : on peut les résoudre en énumérant toutes les possibilités puis en vérifiant, mais l'énumération est d'ordre exponentiel ou factoriel alors que l'on recherche un algorithme polynomial.

[2] Voir Rosen, page 222.

Une autre caractéristique commune de ces problèmes est que, bien qu'il soit difficile (ou impossible ?) de fournir un algorithme qui produit une solution en temps polynomial, il est facile de fournir un algorithme qui vérifie en temps polynomial si une solution donnée est valide.

Les problèmes NP-complets : très longs à résoudre, très rapides à vérifier.

Une question à un million ! 1 000 000 $

Le fait qu'à ce jour aucun des problèmes NP-complets n'ait été résolu en temps polynomial nous porte à croire que cela est impossible. Mais l'impossibilité de les résoudre en temps polynomial n'a pas été démontrée. Cette question ouverte constitue d'ailleurs un des 7 problèmes du millénaire pour chacun desquels un million de dollars est offert en récompense par le Clay Mathematics Institute of Cambridge.

Cela ne veut pas dire qu'il faille abandonner toute tentative de résoudre les problèmes NP-complets par des algorithmes efficaces. Par exemple, si le problème en est un d'optimisation, il est peut-être impossible de trouver un algorithme efficace produisant *la* solution optimale mais possible d'en trouver un produisant une solution *très proche* de la solution optimale. Parfois, il sera possible de trouver un algorithme fonctionnant efficacement pour la grande majorité des cas traités et on s'en contentera.

Si vous êtes confrontés à un problème et que vous établissez qu'il est NP-complet, vous saurez qu'il vaudra mieux chercher une solution approximative. À moins de révolutionner l'informatique et de devenir millionnaire !

Références

1. K. H. Rosen, *Discrete Mathematics and its applications, 5^e édition*, sections 2.1 à 2.3, 2003.
2. Pierre Wolper, *Introduction à la calculabilité, 2e édition,* chapitre 8, 1991. QA76.9 M35 W65 2001
3. Une introduction à la complexité : efficacité des algorithmes, classes des problèmes P, NP et NP-complets, théorème de Cook sur la NP-complétude du problème SAT (1971).
   `http://users.forthnet.gr/ath/kimon/CC/CCC4b.htm`
4. Des articles en français de l'encyclopédie Wikipédia sur la notation grand O, sur la théorie de la complexité, sur la complexité algorithmique : `http://fr.wikipedia.org/wiki/Accueil`
5. Au sujet de la récompense de 1 000 000 $: `http://www.claymath.org/millennium/P_vs_NP/`