Complexité temporelle d'algorithmes : rappels

La complexité en temps $C(\mathtt{A}, I)$ d'un algorithme $\mathtt{A}$ pour résoudre une instance (une entrée) I d'un problème est mesurée comme le nombre d'opérations élémentaires effectuées pour obtenir la solution au problème.

Reste alors à définir ce que l'on entend par opérations élémentaires : il s'agit en général des opérations arithmétiques de base, des tests et des affectations.

On cherche à exprimer la complexité en temps d'un algorithme en fonction de la taille de l'instance à résoudre [1], de manière à avoir une idée du comportement de l'algorithme appliqué à de grandes instances.

$\mathtt{A}$ étant un algorithme pour résoudre un problème Π, on définit la complexité au pire des cas de $\mathtt{A}$: $C_P(\mathtt{A}, n) = \max_{I||I|=n} \{C(\mathtt{A}, I)\}$

Il est bien sûr intéressant de calculer cette complexité de manière exacte. Cependant, il est généralement suffisant de connaître le terme dominant de cette complexité (voire un ordre de grandeur) pour pouvoir apprécier l'évolution du temps d'exécution de l'algorithme quand la taille des instances à résoudre croît.

En effet, les comportements des fonctions sont très différents pour des instances relativement grandes. Ainsi, si l'on a une complexité en $n^3 + 10n$, nous pouvons l'assimiler à une complexité en n^3.

Entre un algorithme de complexité $4n^2 + 10n$ et un de complexité $n^3/2 + 3$, on suggère donc de privilégier le premier.

Taille / Complexité	n	$n \log_2(n)$	n^2	n^3	2^n	n !
$n = 10^2$	0.1 ms	0.6 ms	10 ms	1 s	4.10^6 ans	trop long...
$n = 10^4$	10 ms	0.1 s	100 s	11.5 jours	trop long...	trop long...
$n = 10^6$	1 s	19.9 s	11.5 jours	31.7×10^3 ans	trop long...	trop long...

FIGURE 1 – Temps d'exécution d'un algorithme de complexité donnée (en nombre d'instructions) sur une entrée de taille donnée sur un ordinateur exécutant 10^{-6} instructions par seconde.

Pour pouvoir exprimer précisément ces grandeurs, nous rappelons les notations mathématiques correspondantes.

Ordre de grandeur

Soit f et g deux fonctions de $\mathbb{N}^*$ dans $\mathbb{N}^*$. Alors :

1. $f = O(g)$ s'il existe un réel $c > 0$ et un entier N tel que pour tout $n \geq N$, $f(n) \leq cg(n)$;
2. $f = \Omega(g)$ s'il existe un réel $c > 0$ et un entier N tel que pour tout $n \geq N$, $f(n) \geq cg(n)$;
3. $f = \Theta(g)$ si $f = O(g)$ et $g = O(f)$;

Par exemple :

$$\frac{n^3}{3} + 10n + \log(n) \begin{cases} = O(n^3) \\ = \Theta(n^3) \\ = \Omega(n^3) \\ = O(n^4) \end{cases}$$

1. La taille d'une instance est l'espace mémoire requis pour la représenter. Notons que bien souvent, on considère dans ces calculs d'autres paramètres 'représentatifs' de la taille de l'instance, mais plus simples ou plus parlants. Par exemple, si l'on a un tableau de n nombres entre 1 et 1000, il est plus commode de prendre comme paramètre n que le nombre de bits effectifs utilisés pour coder l'instance.

Le calcul de la complexité d'un algorithme est une composante majeure en vue du choix d'une algorithme pour la résolution d'un problème effectif.

Voici quelques remarques à propos de ce calcul de complexité :
— Le calcul que l'on effectue est souvent une *majoration* de la complexité effective. Le calcul exact est en effet généralement complexe, notamment car le nombre d'opérations dépend de l'instance.
— On s'intéresse généralement au *terme dominant* de cette complexité. Au mieux, on peut trouver un équivalent, mais on cherche souvent simplement un *ordre de grandeur*.
— Etant donné que le calcul est souvent un majorant, si l'on aboutit à un résultat du type $C(n) = O(n^2)$, on peut se demander si le calcul du majorant a été bien fait. La complexité réelle est-elle en réalité en $\Theta(n)$? Pour répondre à cette question, on cherche à construire des instances particulières sur lesquelles l'exécution de l'algorithme requiert bien $\Omega(n^2)$ opérations. On a alors $C(n) = O(n^2)$ et $C(n) = \Omega(n^2)$, donc $C(n) = \Theta(n^2)$.
— Pour simplifier les calculs, on se limite parfois à certaines opérations particulières (le nombre de comparaisons par exemple). Les opérations sur les indices de boucle par exemple sont souvent négligés.

D'un point de vue du vocabulaire, on distingue en particulier les complexités suivantes :
— Complexité logarithmique : $C(n) = \Theta(\log(n))$ (ex : recherche dichotomique dans un tableau trié contenant n éléments).
— Complexité linéaire : $C(n) = \Theta(n)$ (ex : recherche dans un tableau non trié)
— Complexité quasi-linéaire : $C(n) = \Theta(n \log(n))$ (ex : tri par fusion)
— Complexité quadratique : $C(n) = \Theta(n^2)$ (ex : tri par insertion)
— Complexité polynomiale : $C(n) = \Theta(n^a)$, pour un certain $a \geq 1$
— Complexité exponentielle : $C(n) = \Theta(a^n)$, pour un certain $a > 1$. (ex : tours de Hanoï)

Remarque : la complexité d'un algorithme dépend souvent des structures de données que l'on utilise pour résoudre le problème et pour coder l'instance. Les opérations élémentaires dépendent en toute rigueur elles aussi de la taille des nombres (une multiplication entre deux nombres de 1000 bits ne prendra pas le même temps qu'une muiltiplication entre deux nombres de 3 bits).

Tout cela est formellement défini par l'intermédiaire des machines de Turing : la complexité d'un algorithme (d'une machine de Turing) est le nombre maximal d'étapes (de déplacements) de la machine de Turing pour toute entrée de taille n (pour laquelle il y a n symboles sur le ruban).

1 Référence pour en savoir plus

Introduction à l'algorithmique. Cormen, Leiserson, Rivest, Stein. Chapitre 3 (livre disponible en plusieurs exemplaires à la BU).