COMPLEXITÉ

1 Comparaison entre les fonctions

1.1 Definitions

Soit f, g deux fonctions de $\mathbb{N}$ dans $\mathbb{R}_+$.
On note $f = O(g)$ ou $f(n) = O(g(n))$ si et seulement si il existe un entier $K \in \mathbb{R}$ tel que $f(n) \leqslant Kg(n)$ pour tout n assez grand.

$$f = O(g) \iff \exists K \in \mathbb{R}\ \exists n_0 \in \mathbb{N}\ \forall n \geqslant n_0 \quad f(n) \leqslant Kg(n)$$

On note $f = \Theta(g)$ si et seulement si $f = O(g)$ et $g = O(f)$.

Proposition 1 *Quelques exemples fondamentaux : soit α, β deux réels strictement positifs, alors*

- ▷ *si $f(n) = O(n^\alpha)$ et $g(n) = O(n^\alpha)$, alors $f(n) + g(n) = O(n^\alpha)$*
- ▷ *si $f(n) = O(n^\alpha)$ et $g(n) = O(n^\beta)$ et $\alpha < \beta$, alors $f(n) + g(n) = O(n^\beta)$*
- ▷ *si $f(n) = O(n^\alpha)$, alors $nf(n) = O(n^{\alpha+1})$*
- ▷ *si $f(n) = O(n^\alpha)$, alors $\displaystyle\sum_{k=1}^{n} f(k) = O(n^{\alpha+1})$*

2 Comparaison des algorithmes

Même si un algorithme termine, il est possible que la durée du calcul dépasse notre patience. Pour résoudre un certain problème, il est donc nécessaire de comparer les algorithmes et de pouvoir décider si l'un est plus rapide (plus efficace) que les autres. Bien sûr, cette comparaison doit être indépendante de la machine qui va exécuter l'algorithme et ne dépendre que de l'algorithme lui-même et de ses données.

2.1 Taille d'un objet

On considère des algorithmes qui ont un paramètre v appartenant à un ensemble E. On appelle fonction taille sur E toute application de E dans $\mathbb{N}$. Dans la pratique, cette fonction représente la taille de l'objet v : plus il est sensé être gros, plus sa taille est grande. La fonction taille est donc choisie en fonction du problème selon ce qu'on considère comme l'information pertinente.

Exemples

a) Si v est un entier naturel, la taille de v est souvent v lui-même, mais ça peut être aussi le nombre de chiffres de son écriture en base 2 ;

b) si v est un tableau ou une liste, la taille de v est souvent sa longueur ;

c) si v est un couple d'entiers (m, n), la taille peut être $m + n$ ou mn.

2.2 Complexité d'un calcul

Soit E un ensemble pour lequel on a choisi une fonction taille T. On appelle opération élémentaire sur les objets de E toute opération dont le temps d'exécution ne dépend pas de la taille de l'objet.

Exemples

a) si la taille d'un entier (écrit en base 2) est lui-même, la mesure de sa parité est élémentaire : elle consiste à lire le dernier chiffre de son écriture en base 2

b) si la taille d'un tableau est sa longueur, lire ou écrire dans une case du tableau est élémentaire

c) si la taille d'une liste est sa longueur, décomposer la liste en tête et queue ou ajouter un élément en tête sont des opérations élémentaires

Soit A un algorithme de paramètre v appartenant à E. La complexité (ou coût) du calcul de $A(v)$, notée $c_A(v)$ est le nombre d'opérations élémentaires effectuées pour réaliser le calcul.

2.3 Complexité d'un algorithme

Avec les mêmes notations, pour $n \in \mathbb{N}$, on note $E_n = \{v \in E \ / \ T(v) = n\}$.
On appelle complexité de l'algorithme A la fonction C_A de $\mathbb{N}$ dans $\mathbb{N}$ définie par :

$$C_A(n) = \max_{v \in E_n} c_A(v)$$

Cette fonction est aussi appelée complexité dans le pire cas.
On appelle complexité dans le meilleur cas de l'algorithme A la fonction C_A^+ de $\mathbb{N}$ dans $\mathbb{N}$ définie par :

$$C_A^+(n) = \min_{v \in E_n} c_A(v)$$

On appelle complexité moyenne de l'algorithme A la fonction C_A^m de $\mathbb{N}$ dans $\mathbb{R}$ définie par :

$$C_A^m(n) = \sum_{v \in E_n} p(v) c_A(v)$$

où p désigne une probabilité sur E_n.

Remarque. Si E_n est fini, on prend souvent pour p la probabilité uniforme $p(v) = \dfrac{1}{\text{card } E_n}$

2.4 Échelle de complexité

En pratique, on peut difficilement calculer ces complexités de manière exacte. Mais ce qui est aisé est de les comparer à des fonctions classiques.

On dit que A est de complexité ... :
- logarithmique si $C(n) = \Theta(\ln n)$
- linéaire si $C(n) = \Theta(n)$
- quasi-linéaire si $C(n) = \Theta(n \ln n)$
- quadratique si $C(n) = \Theta(n^2)$
- polynomiale si $C(n) = \Theta(n^\alpha)$, où α est un exposant au moins égal à 1
- exponentielle si $C(n) = \Theta(z^n)$, où z est un réel strictement supérieur à 1
- sur-exponentielle si pour tout $z > 1$, $z^n = o(C(n))$

De plus, il est en général difficile de montrer l'égalité $C(n) = \Theta(g(n))$: on se contente dans la plupart de cas d'une estimation en $O(g(n))$, de sorte de la fonction dominante g soit raisonnable, c'est-à-dire la plus petite possible avec les moyens utilisés (rappelons qu'on cherche l'algorithme le plus rapide).

En pratique, un algorithme de complexité logarithmique est considéré comme très bon (recherche dans un tableau trié), de complexité linéaire ou quasi-linéaire comme bon (tri fusion), et acceptable s'il est de complexité polynômiale avec un exposant inférieur ou égal à 3 (tri par sélection). Au-delà, le temps de calcul devient souvent un obstacle à l'utilisation de l'algorithme, sauf si n est « petit » (résolution de sudoku).

Avec un ordinateur personnel travaillant à 2 Ghz, on obtient les temps de calcul suivants selon le type de complexité :

nbr. d'op. élém. n	$\ln n$	n	$n \ln n$	n^2	n^3	2^n
100	$2,3.10^{-9}$ s	5.10^{-8} s	$2,3.10^{-7}$ s	5.10^{-6} s	5.10^{-4} s	13000 u
1000	$3,4.10^{-9}$ s	5.10^{-7} s	$3,4.10^{-6}$ s	5.10^{-4} s	0,5 s	
10^4	$4,6.10^{-9}$ s	5.10^{-6} s	$4,6.10^{-5}$ s	0,05 s	8 m 20 s	
10^5	$5,7.10^{-9}$ s	5.10^{-5}s	$5,7.10^{-4}$ s	5 s	5 j 18 h	
10^6	$6,9.10^{-9}$ s	5.10^{-4} s	$6,9.10^{-3}$ s	8 m 20 s	16 a	
10^9	1.10^{-8} s	0,5 s	10 s	16 a	16.10^9 a	