Complexité

Motivation

On cherche à modéliser les ressources dont a besoin un programme :

- temps
- mémoire

- en moyenne
- dans le pire des cas

Permet de comparer différents algorithmes pour une même spécification

Rappel de notations

f et g fonctions de $\mathbb{N}$ dans $\mathbb{R}^+$

- On note $f = \mathcal{O}(g)$ s'il existe une constante K et un rang N tel que pour tout $n \geq N$ on a $f(n) \leq Kg(n)$.
  C'est une relation de préordre (réflexive, transitive).
- On note $f = \Theta(g)$ si $f = \mathcal{O}(g)$ et $g = \mathcal{O}(f)$.

Complexité en temps

Pour modéliser la complexité en temps d'un programme, on va borner le nombre d'opérations élémentaires (comparaison, opération arithmétique de base, ...) en fonction de la taille de l'entrée.

Exemple : le tri rapide (quick sort)

On cherche à trier un tableau :

- on choisit un pivot $p = t[i]$ dans le tableau ;
- on calcule les deux tableaux $t_{<p}$ et $t_{>p}$ formés respectivement des éléments plus petits que p et plus grands que p ;
- on trie récursivement ces tableaux ;
- on renvoie le tableau formé de $t_{<p}\ p\ t_{>p}$.

Complexité du tri rapide

Soit $R(n)$ la complexité du tri rapide, où n est la taille du tableau, en fonction du nombre de comparaisons.
Pour créer les sous-tableaux $t_{<p}$ et $t_{>p}$ on a besoin de $n-1$ comparaison (de chaque élément autre que p avec p).
Soit $n_{<p}$ et $n_{>p}$ la taille de ces sous-tableaux, on a $n = n_{<p} + n_{>p} + 1$.
On a donc la relation de récurrence

$$R(n) = n - 1 + R(n_{<p}) + R(n_{>p})$$

Si on suppose que $n_{<p} = n_{>p} = n/2$ (ce qui sera le cas en moyenne) alors on a

$$R(n) = n - 1 + 2R(n/2)$$

Résolution de la récurrence

$$R(n) = n - 1 + 2R(n/2)$$

On pose $T(k) = R(2^k) - 1$

$$\begin{aligned} T(k) &= 2^k - 1 + 2R(2^k/2) - 1 \\ &= 2^k + 2(R(2^{k-1}) - 1) \\ &= 2^k + 2\,T(k-1) \end{aligned}$$

$$T(k) = 2^k + 2\,T(k-1)$$

Suite récurrente linéaire non homogène. La partie homogène vérifie :

$$U(k) = 2\,U(k-1)$$

ce qui se résoud en $U(k) = 2^k$.
On pose $V(k) = T(k)/2^k$. On a

$$\begin{aligned} V(k) &= \frac{2^k + 2\,T(k-1)}{2^k} \\ &= 1 + T(k-1)/2^{k-1} \\ &= 1 + V(k-1) \end{aligned}$$

ce qui se résoud en $V(k) = k$.
On a donc $T(k) = 2^k\,V(k) = k2^k$. Par conséquent $R(n) = T(\log_2(n)) + 1 = \mathcal{O}(n \log n)$.

Remarques

- Pour un tri, on ne peut pas espérer mieux que $\mathcal{O}(n \log n)$
- Dans le pire des cas (pivot toujours mal choisi), le tri rapide est en n^2
- Ici, on a pu donner une complexité exacte. En pratique, on doit souvent faire des approximations.

Classe de complexité

- $\mathcal{O}(\log n)$ logarithmique
- $\mathcal{O}(n)$ linéaire
- $\mathcal{O}(n \log n)$ quasi-linéaire
- $\mathcal{O}(n^2)$ quadratique
- $\mathcal{O}(n^k)$ (k constante) polynomiale
- 2^n exponentielle

Comparaison

$T(n)$	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$	$n = 60$
$\log n$	1μs	1,3μs	1,5μs	1,6μs	1,7μs	1,8μs
n	10μs	20μs	30μs	40μs	50μs	60μs
$n \log n$	10μs	26μs	44μs	64μs	85μs	107μs
n^2	100μs	400μs	900μs	1,6ms	2,5ms	3,6ms
n^5	0,1s	3s	24s	1,7min	5min	13min
2^n	1ms	1s	18min	13j	36a	36 600a

Classe de problèmes

Un problème est dit :

- décidable s'il existe un algorithme qui permet de le résoudre
- indécidable s'il n'existe pas de tel algorithme (exemple : problème de l'arrêt)
- dans P s'il existe un algorithme (déterministe) polynomial qui permet de le résoudre
- dans NP s'il existe un algorithme non-déterministe polynomial (équivalent à dire qu'on peut vérifier une solution du problème en temps polynomial)
- NP-complet si la résolution de n'importe quel problème NP peut se ramener à la résolution de ce problème particulier

On ne sait pas si P = NP (1M$!)