Travaux pratiques

Complexité

Complexité temporelle

Un même problème pourra donner lieu à la création de divers algorithmes. Faire tourner le programme issu d'un algorithme se fait en un temps qui dépend de l'état de la machine, du réseau, etc. Il reste au programmeur à étudier l'impact du choix de l'algorithme sur le temps pris. On considère que ce temps de sortie dépend de la **taille** de l'entrée, donnée généralement spécifiée ou conforme à l'intuition : la taille d'une liste sera par exemple le nombre de données qu'elle contient.

Supposons qu'on dispose de deux algorithme A et B, lesquels traitent un problème dont l'entrée est de taille n respectivement en temps $3 \times n$ ou $12000 \times n$. Le second algorithme est plus lent que le premier, mais il sera toujours préférable *pour de très grandes entrées* à l'algorithme C qui propose une solution en temps $2 \times n^2$.
Les complexités usuellement utilisées sont répertoriées dans le tableau suivant :

Dénomination	Définition
logarithmique	$\Theta(\log(n))$
linéaire	$\Theta(n)$
quasi linéaire	$\Theta(n \log(n))$
polynomiale	$\Theta(n^k)$, avec $k > 0$
exponentielle	$\Theta(a^n)$, avec $a > 1$

Exemples d'algorithmes de complexité donnée

- Constante : accession à la tête d'une liste, ou à une case de tableau
- Logarithmique : recherche dans un arbre, recherche dichotomique dans un tableau
- Linéaire : Parcours d'une liste ou d'un tableau (recherche d'un élément, max, min), produit scalaire
- Quadratique : Produit matrice vecteur, double parcours d'un tableau (tri par sélection)
- Exponentielle : énumération de sous-ensemble, de permutations, recherche de mot de passe par force brute

Conséquence de la taille n de l'entrée sur le temps de traitement $T(n)$

- Constante : modifier la taille de l'entrée ne modifie pas T
- Logarithmique : il existe une constante k, telle que $T(2 \times n) = T(n) + k$. Ainsi, $T(2^n) = T(1) + 2k$
- Linéaire : Multiplier la taille de l'entrée par 2 multiplie T par 2 (et ainsi de suite !)
- Quadratique : Multiplier la taille de l'entrée par k multiplie T par k^2 (et ainsi de suite !)
- Exponentielle : Ajouter k à la taille de l'entrée multiplie par e^k le temps T. Le temps devient très grand dès qu'on augmente un peu la taille de l'entrée. Par exemple : supposons qu'un algorithme prenne un temps d'une seconde pour une entrée de taille 1. Le même algorithme mettra un peu plus de 8000 secondes pour traiter une entrée de taille 10 (un peu plus de deux heures) ; et environ 10^{43} secondes pour une entrée de taille 100. A titre de comparaison, il s'est écoulé moins de 5.10^{17} secondes depuis la naissance de l'univers.

Conclusion : quand on le peut, on évite les algorithmes de complexité exponentielle !

Complexité en espace

Il est parfois pertinent d'observer les ressources mémoire utilisées pour un calcul : si un algorithme utilise une quantité exponentielle de mémoire en fonction de la taille de l'entrée, même s'il est très rapide, il ne sera pas utilisable pour de grandes entrées, en raison des limitations physiques des ordinateurs.
La complexité spatiale est donnée par le nombre maximal de cases mémoires utilisées en même temps **en plus de celles mobilisées par la donnée en entrée** pour effecteur un calcul. Si on réutilise n cases m fois, la complexité spatiale reste égale à n.
Exemples

- Trouver le premier élément d'un tableau : constant, **quelle que soit la taille du tableau**, on n'utilise aucun espace supplémentaire.
- Renvoyer une matrice carrée dont chaque ligne est le tableau d'entrée : espace quadratique

Différents types de complexité

La complexité d'un algorithme peut dépendre de l'entrée, même à taille fixée. Dans ce cas, on pourra spécifier le type de complexité temporelle observé : est-ce dans le pire des cas ? en moyenne ? dans le meilleur cas ? Par exemple, l'algorithme du tri à bulles dans le meilleur cas (liste déjà triée) est linéaire, dans le pire cas quadratique (liste inversée). On s'intéressera le plus souvent aux complexités en moyenne et dans le pire cas.

Les tris

- tri à bulles : quadratique (moyenne et pire cas)
- tri à bulles : insertion (moyenne et pire cas)
- tri à bulles : sélection (moyenne et pire cas)
- tri fusion : quasi-linéaire (moyenne et pire cas)
- tri rapide : quasi linéaire (moyenne), quadratique (pire cas)

La complexité étant une donnée portant sur les performances asymptotiques, on s'intéresse à **l'ordre de grandeur**, qui peut cacher une constante éventuellement très grande. Dans le cas des petites entrées, on pourra préférer utiliser un algorithme asymptotiquement moins performant. Pour le tri rapide, on considère ainsi que les appels récursifs se font tant que la taille du tableau à trier excède 15 (environ). En dessous, on utilise un tri par insertion ou par sélection, plus efficaces sur de petits effectifs.

exemple

L'addition naïve sur des grands nombres de même taille n comporte opérations élémentaires, l'algorithme est donc

La multiplication naïve d'un grand nombre par un chiffre k comporte opérations élémentaires, l'algorithme est donc

La multiplication naïve sur des grands nombres de même taille n comporte opérations élémentaires, l'algorithme est donc

L'algorithme de Karatsuba découpe le problème en sous-problèmes de taille environ deux fois moindres.
Ainsi $K(n) \leq \alpha K(\lceil \frac{n}{2} \rceil) + \beta \frac{n}{2} + \gamma$ avec α, β, γ des constantes (plus précisément, $\alpha = 3$).
Notons $u_n = K(2^n)$: $u_{n+1} \leq \alpha u_n + \beta' 2^n + \gamma$
Posons $v_n = \frac{u_n}{\alpha^n}$. La relation de récurrence précédente devient :

$v_{n+1} \leq v_n + \dfrac{\beta' 2^n + \gamma}{\alpha^{n+1}}$, ce dont on déduit
$\forall n \in \mathbb{N}, v_n \leq \displaystyle\sum_{k=0}^{n-1} \frac{\beta' 2^k + \gamma}{\alpha^{k+1}}$ qui est une quantité bornée lorsque $\alpha = 3$ (série géométrique), ce dont on déduit qu'il existe un réel (positif) A, tel que pour tout n, $u_n \leq A \times 3^n$.

Le résultat semble borné par une quantité exponentielle, ce qui n'est pas ce qu'on cherchait, mais il faut revenir à $u_n = K(2^n)$. Ainsi, pour n une puissance de 2, $K(n) \leq A3^{\log_2(n)} = A3^{\ln n \log_2 e} = An^{\ln 3 \log_2 e}$, avec $\ln 3 \log_2 e \leq 1,6$.

Qu'en penser ?

Exercices

1. Estimer la complexité temporelle (au papier et en machine, donc avec génération d'objets de taille en fonction de leur taille) des algorithmes suivants :
   - calcul l'indice du l'élément maximal d'un tableau de taille n
   - calcul du second maximum
   - somme des éléments d'un tableau de n lignes et n colonnes
   - retournement horizontal d'une image
   - calcul de l'ensemble des permutations de l'ensemble $\{1, ..., n\}$
2. Si un algorithme est de complexité spatiale $S(n)$, peut-on trouver un minorant de sa complexité temporelle ?
3. Vérifier en machine l'ordre de grandeur des tris.
4. Vérifier la partie de l'exemple concernant la multiplication rapide de Karatsuba en machine.