Algorithmes et complexité temporelle

Pourquoi introduire la notation "Grand Theta" $\Theta(\cdot)$?

Voici un exemple pratique : supposons que pour résoudre un problème donné, vous ayez écrit un algorithme dont le temps d'exécution est de 1 minute avec des données d'entrée de taille n. Vous aimeriez bien savoir ce qu'il va se passer si vous doublez la taille des données d'entrée de votre algorithme : celui-ci prendra-t-il le double de temps, moins que ça ou plus que ça ?

- Si votre algorithme a besoin d'effectuer environ n opérations pour obtenir un résultat avec des données d'entrée de taille n, alors effectivement, le temps d'exécution de celui-ci sera logiquement doublé avec des données d'entrée de taille $2n$, donc votre algorithme s'exécutera en 2 minutes.

- Si par contre votre algorithme, avec des données d'entrées de taille n, a besoin d'effectuer environ n^2 opérations pour obtenir un résultat, alors pour des données d'entrée de taille $2n$, on obtient :

si n^2 opérations = 1 minute, alors $(2n)^2$ opérations = $4n^2$ opérations = 4 minutes : le temps d'exécution est quadruplé !

- Que se passe-t-il maintenant si votre algorithme a besoin d'effectuer environ $2n$ opérations pour obtenir un résultat avec des données d'entrées de taille n ? En reproduisant le calcul ci-dessus, on obtient :

si $2n$ opérations = 1 minute, alors $2(2n)$ opérations = 2 minutes : le temps d'exécution est doublé, exactement comme dans le cas où environ n opérations sont nécessaires avec des données d'entrée de taille n.

En conclusion, peu importe de savoir en pratique si, avec des données d'entrées de taille n, le nombre d'opérations à effectuer est n, $2n$, $3n\ldots$ Par contre, une grande différence existe entre n et n^2. C'est ce que la notation $\Theta(\cdot)$ permet de retenir en s'affranchissant des constantes.

Exercice : Si le temps d'exécution d'un algorithme avec des données d'entrée de taille n est de 1 minute, si $n = 1'000$ et si la complexité temporelle de l'algorithme est environ $\log_2(n)$, de combien de temps l'algorithme aura-t-il besoin pour traiter des données d'entrée de taille $2n$? Même question si la complexité temporelle de l'algorithme est environ 2^n ? Attention, les réponses à ces deux questions risquent de vous surprendre ! (voir plus bas)

Petites valeurs de n

Il importe de bien garder en mémoire que toutes les complexités temporelles d'algorithmes calculées dans ce cours n'ont de sens que pour de relativement grandes valeurs de n (vu la définition même de la notation $\Theta(\cdot)$). Pour de petite valeurs de n, certains algorithmes très peu efficaces à grande échelle peuvent se révéler tout à fait adéquats (comme par exemple essayer toutes les possibilités de voyage lorsqu'on doit se rendre en train dans 3 villes différentes).

Réponses à l'exercice ci-dessus :

- Dans le cas où la complexité temporelle est environ $\log_2(n)$, on obtient :

$\log_2(1'000) \simeq 10$ opérations = 1 minute, donc chaque opération prend environ 6 secondes dans ce cas, et $\log_2(2'000) = \log_2(2) + \log_2(1'000) \simeq 11$ opérations = 1 minute et 6 secondes.

- Dans le cas où la complexité temporelle est environ 2^n, on obtient par contre : $2^{1'000}$ opérations = 1 minute, donc $2^{2'000}$ opérations = $2^{1'000}\, 2^{1'000}$ opérations = $2^{1'000}$ minutes $\simeq 10^{300}$ minutes !

Et finalement : un détail à propos des boucles “pour” et “tant que”

Les boucles simples “Pour i allant de 1 à n” sont en fait toujours implémentées de manière concrète dans un ordinateur comme des boucles conditionnelles “Tant que”. En effet, on peut toujours réécrire une boucle simple au moyen d'une boucle conditionnelle de la manière suivante :

Algorithme

entrée : ...
sortie : ...

Pour *i allant de* 1 *à* n
    instruction dépendant de i

Sortir : ...

Algorithme

entrée : ...
sortie : ...

$i \longleftarrow 1$
Tant que $i \leq n$
    instruction dépendant de i
    $i \longleftarrow i + 1$

Sortir : ...

Dès lors, pourquoi utiliser des boucles simples dans un algorithme ? Tout simplement parce qu'elles sont plus faciles à lire pour nous, à cause de leur notation plus compacte ! Mais de manière concrète, une machine interprétera toujours l'algorithme de gauche par sa version de droite, plus détaillée.

On voit encore une fois ici l'intérêt d'introduire la notation $\Theta(\cdot)$: en supposant qu'un nombre fixe d'instructions soient exécutées au coeur de chacune de ces deux boucles, on déduira que la complexité de chacune de celles-ci est $\Theta(n)$. Pas besoin d'aller compter le nombre exact d'opérations effectuées (qui au final est le même dans les deux cas, mais n'est pas forcément facile à compter avec la notation compacte à gauche).