Complexité algorithmique

Par Dimitri PIANETA

Septembre 2018

1) Définitions :

Algorithme : Procédure de calcul bien définie qui prend en entrée une valeur, ou un ensemble de valeurs, et qui donne en sortie une valeur, ou un ensemble de valeurs.

Complexité : est le nombre d'opérations élémentaire qu'il doit effectuer pour mener à bien un calcul en fonction de la taille des données d'entrée.

2) La taille des données :

La taille des données va dépendre du codage de ces entrées. On choisit comme taille la ou les dimensions les plus significatives.

3) Le temps de calcul :

Le temps de calcul d'un programme dépend de plusieurs éléments :

- la quantité de données bien sûr ;

- mais aussi de leur encodage ;

- de la qualité du codage engendre par le compilateur ;

- de la nature et la rapidité des instructions du langage ;

- de la qualité de la programmation ;

- et de l'efficacité de l'algorithme.

4) Notions mathématiques

4.1 – Notation de Landau ou O

4.1.1 Définition

Remarque

La notation O est utilisée pour indiquer la borne supérieure asymptotique.

Définition

Soit g une fonction définie sur une partie des nombres réels. L'ensemble O(g) est défini ainsi :

O(g) = { f fonction définie sur une partie de $\mathbb{R}\exists S > \quad 0, \exists n_0 \in \mathbb{N}$ tel que $\forall x \geq n_0, |f(x)| \leq c|g(x)|\}$

Nous disons « grand o » de g pour O(g).

5) Classement

Complexité	**Description**
Complexité *O(1)* Complexité **constante** ou temps contant	L'exécution ne dépend pas du nombre d'éléments en entrée mais s'effectue toujours en un nombre constant d'opérations.
Complexité *O(log(n))* Complexité **logarithmique**	La durée d'exécution croit légèrement avec n. Ce cas de figure se rencontre quand la taille du problème est divisée par une entité constante à chaque itération.
Complexité *O(n)* Complexité l**inéaire**	C'est typiquement le cas d'un programme avec une boucle de 1 à n et le corps de la boucle effectue un travail de durée constante et indépendante de n.
Complexité O(nlog(n)) Complexité **n-logarithmique**	Se rencontre dans les algorithmes où à chaque itération le taille du problème est divisée par une constante avec à chaque fois un parcours linéaire des données. Un exemple typique de ce genre de complexité est l'algorithme de tri « quick sort » qui de manière récursive, divise le tableau à trier en deux morceaux par rapport à une valeur particulière appelée pivot, trie ensuite la moitié du tableau puis l'autre moitié… S'il n'y a pas cette opération de « division » l'algorithme serait logarithmique puisqu'on divise par 2 la taille du problème à chaque étape. Mais, le fait de reconstituer à chaque fois le tableau en parcourant séquentiellement les données ajoute ce facteur n au log(n). Noter que la complexité n-logarithmique est tributaire du bon choix du pivot.
Complexité O(n^2) Complexité **quadratique**	Typiquement c'est le cas d'algorithmes avec deux boucles imbriquées chacune allant de 1 à n et avec le corps de la boucle interne qui est constant.
Complexité O(n^3) Complexité **cubique**	Idem quadratique mais avec ici un exemple trois boucles imbriquées.
Complexité O(n^p) Complexité **polynomiale**	Algorithme dont la complexité est de la forme O(n^p) pour un certain p.
Complexité O(2^n) Complexité **exponentielle**	Les algorithmes de ce genre sont dits « naïfs » car ils sont inefficaces et inutilisables dès que n dépasse 50.