11

Calculs de complexité

Qu'est-ce que la « complexité » d'un programme ? La complexité en temps d'une fonction (donnée sous la forme d'un algorithme ou d'un programme PYTHON) représente le nombre d'opérations « élémentaires » nécessaires à son exécution en fonction de la taille des données. On parle aussi complexité temporelle ou du temps d'exécution de la fonction.

Il faut préciser ce que l'on entend par taille des données, pour une liste L par exemple il s'agit de son nombre d'éléments.

En général, on donne simplement un ordre de grandeur du nombre d'opérations en utilisant la notation O. L'avantage de cette notion (par rapport à une mesure réelle du temps mis par l'ordinateur pour faire tourner le programme) est qu'elle est indépendante de l'ordinateur, du système d'exploitation et même du langage de programmation (pour un algorithme).

On distingue souvent :

- La complexité dans le pire des cas (nombre maximum d'opérations effectuées pour une donnée de taille n) ;
- La complexité dans le meilleur des cas (nombre minimum d'opérations effectuées pour une donnée de taille n) ;
- La complexité en moyenne que l'on peut définir comme la somme :

$$\sum_{\substack{\text{données } x \\ \text{de taille } n}} P_x \times N_x$$

où P_x est la probabilité d'apparition de la donnée x et N_x est le nombre d'opérations nécessaires pour effectuer l'algorithme sur x. La complexité en moyenne est intéressante mais difficile à expliciter en général (il faut proposer un modèle probabiliste pour l'apparition des données et pouvoir ensuite calculer la somme).

Sauf mention explicite du contraire, on s'intéresse en général aux complexités dans le pire des cas.

Qu'est ce que la complexité en espace ? C'est un ordre de grandeur de l'espace mémoire nécessaire à la fonction pour son exécution. On l'exprime également en utilisant la notation O et en fonction de la taille des données (par contre, on ne comptabilise pas la place qu'il faut pour stocker ces données).

À quoi servent les différentes notions de complexité ? Lorsque l'on dispose de plusieurs algorithmes réalisant la même tâche, les différentes notions de complexité permettent de comparer l'efficacité de ces algorithmes. Cette comparaison n'est en général pas absolue : on pourra dans certaines situations préférer un algorithme lent mais utilisant peu d'espace mémoire à un algorithme rapide utilisant beaucoup d'espace mémoire et dans d'autres cas faire le choix inverse.

Quels sont les ordres de grandeur usuels pour la complexité ? On rencontre souvent les ordres de grandeur suivants :

- $O(n)$ (complexité linéaire) : c'est la complexité d'un algorithme qui procède en examinant chaque composante d'une donnée de taille n (typiquement à l'aide d'une boucle répétée n fois). Exemples : recherche d'un élément dans une liste, recherche du maximum, calcul de la somme des termes d'une liste ;
- $O(n^2)$ (complexité quadratique) : c'est la complexité d'un algorithme qui effectue une opération en temps $O(n)$ *pour chaque composante* d'une donnée de taille n (typiquement lorsqu'il y a deux boucles de taille n imbriquées). On rencontre plus généralement des complexités en $O(n^p)$ (complexité polynomiale) ;
- $O(\ln(n))$ (complexité logarithmique) : c'est la complexité d'un algorithme qui dispose d'un moyen pour diviser la taille du problème par 2 (ou par un facteur constant) à chaque étape. Exemple : la recherche dans une liste triée (recherche d'un nom dans un annuaire) ;
- $O(n\ln(n))$: on verra plus tard que c'est la complexité optimale pour trier un tableau de taille n (programme de deuxième année) ;
- $O(2^n)$ (complexité exponentielle) : c'est la complexité que l'on obtient en général quand on utilise un algorithme qui recherche une solution en examinant tous les cas possibles. Ces algorithmes sont inutilisables en pratique, sauf pour de très petites valeurs de n.

Comment effectuer un calcul de complexité ? En général on s'intéresse à la complexité en temps et dans le pire cas. On retiendra les points suivants.

- En présence d'une boucle, il faut établir combien de passages (au maximum) on fera dans la boucle ;
- Pour les algorithmes ou programmes récursifs (en deuxième année), on établit une relation de récurrence sur la complexité.