Algorithmes et Complexité
2 mars 2006

Ici, on oublie que Maple sait faire plein de choses évoluées. On va utiliser ses fonctions les plus élémentaires pour parler de complexité et les appliquer à des tâches simples.

1 Complexité

D'abord, qu'est-ce qu'un algorithme? C'est la description d'une suite d'opérations qu'on applique mécaniquement à des données d'entrée et qui conduisent au calcul d'une sortie. En Maple, les algorithmes prennent la forme de procédures. Les `if`, `for`, `while`, etc... sont des briques de base pour décrire ces suites d'opérations répétitives trop longues à faire à la main et qu'on délègue à une machine. En théorie de la complexité on étudie la rapidité des algorithmes afin de les comparer et retenir les meilleurs. Améliorer la vitesse d'un algorithme peut être une contribution scientifique majeure et rapporter beaucoup d'argent. Commençons par une définition.

Définition *(complexité d'un algorithme):*
La complexité d'un algorithme est le nombre d'opérations élémentaires qu'il doit effectuer pour mener à bien un calcul en fonction de la taille des données d'entrée.

Par exemple, additionner deux nombres de taille n avec l'algorithme qu'on apprend à l'école nécessite n opérations élémentaires (additions de nombres compris entre 0 et 9, avec éventuellement une retenue). On dira alors que $C(n) = O(n)$. En effet, il est souvent difficile de trouver une formule exacte pour exprimer la complexité et on se contente souvent d'un ordre de grandeur ($O(f(n))$ quand $n \to \infty$).

Des dollars et des nombres premiers

Vous avez peut-être entendu parler de la conjecture $P \neq NP$. Démontrer ce résultat ou l'infirmer rapporte un million de dollar, versés par la fondation Clay (qui a répertorié une dizaine de problèmes théoriques pour la résolution desquels elle s'engage à donner cette somme). P est l'ensemble des problèmes de complexité polynomiale, c'est-à-dire tels qu'il existe k tel que $C(n) = O(n^k)$. En pratique, ce sont les problèmes qu'un ordinateur peut résoudre en un temps raisonnable. NP est en gros la classe des problèmes pour lesquels on peut

vérifier en temps polynomial qu'une solution est correcte. On a $P \subset NP$ mais on n'a pas forcément égalité. Prenons un exemple pour donner une idée de ce que ça signifie. La question "Est-ce que 53308290611 est composite?" demande de trouver un diviseur de ce nombre, ce qui peut être très long. Par contre si quelqu'un dit "Oui, parceque 224737 divise 53308290611", vérifier cette information est beaucoup plus aisé.
Une histoire intéressante à ce propos. Jusqu'en 2003, la question de NP "Est-ce que ce nombre est premier?" était de complexité exponentielle, c'est-à-dire que $C(n)$ était de la forme $O(a^n)$. En pratique, "complexité exponentielle" signifie que le temps mis par l'algorithme pour résoudre le problème est beaucoup trop long et qu'on ne peut pas envisager d'utiliser cet algorithme pour faire un grand nombre de calculs (imaginez que chaque calcul prend 10 ans, vous n'avez pas envie d'en faire 100...). Depuis 2003, il existe un algorithme polynômial pour résoudre la question de la primalité, ce qui a redonné pas mal d'espoir aux défenseurs de $P = NP$. Ceux-ci pensent que tout problème NP peut être résolu en temps polynômial, et qu'on a juste pas encore été assez malins pour trouver le bon algorithme (10%). Les défenseurs de $P \neq NP$ pensent que certains problèmes de NP, sont impossibles à résoudre en temps polynômial (60%).

2 La multiplication

Voici une des opérations de base les plus importantes de l'informatique que l'on voudrait traîter le plus efficacement possible. Si nous n'avions pas les chiffres arabes, et que nous étions restés aux chiffres romains, notre algorithme de multiplication serait très lent et compliqué. Grâce à notre système positionnel, on a un algorithme assez concis (celui qu'on voit à l'école primaire) pour faire des multiplications. Décrivons-le sur deux nombres de taille n. On parcours un des nombres de droite à gauche et on multiplie chacun de ses chiffres par l'ensemble des chiffres de l'autre nombre, parcouru également de droite à gauche. On fait ainsi n^2 opérations élémentaires (multiplication de nombres compris entre 0 et 9 avec éventuellement retenue). A la fin il faut faire n additions, qui sont négligeables devant des multiplications. On a donc $C(n) = O(n^2) + n \quad o(1) = O(n^2) + o(n) = O(n^2)$. On dit que l'algorithme classique de multiplication est "en n^2".
Peut-on faire mieux? Oui. L'algorithme du mathématicien tchétchène Karatsuba (!) fait les multiplications en $O(n^{\ln(3)/\ln(2)}) \simeq O(n^{1.58})$. C'est un gain substantiel car on fait environ *sqrtn* fois moins d'opérations. L'algorithme de Karatsuba repose sur le principe "diviser pour régner". On coupe chaque opérande de la multiplication en deux et on réorganise les termes astucieusement. Voici le principe:

$$(a \cdot 10^k + b)(c \cdot 10^k + d) = ac \cdot 10^{2k} + (ac + bd - (a - b)(c - d)) \cdot 10^k + bd$$

3 Calcul de x^n

Ecrivez une procédure pour faire le calcul de x^n. Avez-vous une idée pour l'améliorer? Calculez les complexités des différentes solutions.