L'infini et l'univers des algorithmes

GILLES DOWEK

On ne peut pas toujours prévoir le nombre d'opérations nécessaires à un calcul. Aussi, pour permettre l'expression de tous les algorithmes, il faut introduire la possibilité que les calculs durent infiniment longtemps.

Un algorithme décompose un calcul en des suites d'opérations élémentaires. Quand un algorithme prescrit un nombre infini d'opérations élémentaires, on n'obtient jamais le résultat. Aussi, faut-il tenter de maîtriser ou de chasser cet infini.

Pour multiplier 35 par 12, nous devons multiplier 5 par 2, poser le 0, retenir le 1, multiplier 3 par 2, ajouter la retenue… au bout de nos efforts, nous obtiendrons le résultat : 420. Cette recette qui décrit l'enchaînement des opérations élémentaires d'une multiplication est dénommée l'algorithme de la multiplication, ou plutôt un algorithme de la multiplication, car il existe d'autres méthodes de calcul possibles. Ainsi, 35 + 35 + 35 + 35 + 35 + 35 + 35 + 35 + 35 + 35 + 35 + 35 est une méthode, un algorithme envisageable pour multiplier 35 par 12, à l'aide d'additions successives. Il en existe bien d'autres.

Ces algorithmes sont des méthodes de calcul qui donnent un résultat après un nombre fini d'opérations et dans un temps fini. L'infini ne semble ainsi jouer aucun rôle dans l'univers des algorithmes, et pourtant la notion d'algorithme ne peut être définie sans utiliser l'infini. L'infini est indispensable, mais il faut le maîtriser.

La définition d'un algorithme

Si nous utilisons des algorithmes depuis l'Antiquité et que le mot algorithme vient du nom du mathématicien Al Khwarizmi qui vécut à Bagdad au IXe siècle, ce n'est qu'au XXe siècle que cette notion a été définie avec précision.

Dans les années 1920, Thoralf Skolem (1887-1963) a proposé une première définition de la notion d'algorithme liée à la récurrence. Ce procédé consiste à définir une fonction en donnant sa valeur en 0 et en indiquant comment passer de sa valeur en n à sa valeur en $n + 1$. Par exemple, la fonction e, qui à chaque nombre entier n associe le nombre 2^n : $e(0) = 1$, $e(1) = 2$, $e(2) = 4$, $e(3) = 8$, $e(4) = 16$, … est définie par récurrence, en indiquant que sa valeur en 0 est 1 et qu'on passe d'une valeur à la suivante en la doublant.

Les fonctions qui sont définies par leur valeur en 0, ou en une autre valeur initiale, et par une opération permettant de passer directement d'une valeur à la suivante sont appelées, selon un terme dû à Rosza Péter, récursives primitives. La fonction $e(n) = 2^n$ est récursive primitive. Elle est définie par $e(0) = 1$ et $e(n + 1) = 2e(n)$. Richard Dedekind, qui avait déjà étudié les définitions par récurrence en 1888, avait montré que l'addition, la multiplication, l'élévation à la puissance étaient des fonctions récursives primitives.

La définition d'une fonction par récurrence donne un algorithme pour le calcul des valeurs de cette fonction. Ainsi, pour calculer la valeur de la fonction e en 10, il suffit de calculer sa valeur en 0, puis en 1, en 2,…, en 9 et enfin en 10. On obtient ainsi $e(0) = 1$, $e(1) = 2$, $e(2) = 4$, …, $e(9) = 512$, et $e(10) = 1\,024$.

On programme aisément de telles fonctions récursives primitives en utilisant des boucles *For*. Chaque itération de la boucle exécute une opération élémentaire ; on peut déterminer le nombre d'itérations nécessaires au calcul en fonction de la valeur que l'on souhaite calculer. Si l'on demande au programme de calculer la fonction e pour n égal à 10, après 10 itérations, 1 024 s'affichera sur l'écran de l'ordinateur.