Stages algorithmique	La notion d'algorithme

Nous remercions Christian Vassard de permettre la reproduction d'un article écrit pour une brochure APMEP en 2000*. Le texte originel a été légèrement modifié.*

• Définition

La notion d'algorithme, bien qu'ancienne, n'a été définie et étudiée qu'au XX^{e} siècle, quand il s'est agi pour les mathématiciens de réfléchir aux fondements logiques des mathématiques. Essayons en quelques mots de préciser ce que cette notion recouvre.

Un algorithme peut être défini comme un processus, une suite d'instructions permettant la résolution d'un problème donné dans toute sa généralité et en particulier quelle que soient les données du problème.

Plus précisément, un algorithme doit vérifier les conditions suivantes :
1) il doit pouvoir être écrit dans un certain langage (c'est-à-dire un ensemble de mots écrits dans un alphabet défini) ; sans entrer dans les détails, un tel langage peut ressembler à un langage de programmation comme celui d'une calculatrice ;
2) des données lui sont soumises au départ (ce sont des *entrées*) ;
3) l'algorithme lui-même est un processus s'exécutant étape après étape ;
4) l'action à chaque étape est strictement déterminée par l'algorithme, les entrées et les résultats obtenus dans les étapes précédentes (on notera le caractère strictement *déterministe* de cette action) ;
5) l'algorithme retourne une réponse clairement spécifiée (de préférence celle que l'on attend...), appelée *sortie* ;
6) quelles que soient les entrées, l'exécution doit se terminer en un nombre fini d'étapes.

C'est un concept-clé de l'informatique : l'ordinateur ne peut traiter que les problèmes pour lesquels un algorithme existe ; les langages de programmation, comme le BASIC ou le PASCAL, permettent à l'ordinateur de comprendre l'algorithme qu'on veut lui soumettre.

Cette « définition » étant posée, deux questions viennent à l'esprit, questions auxquelles nous tenterons d'apporter une réponse dans la suite :
– existe-t-il un algorithme pour n'importe quel problème donné ?
– un algorithme doit-il finir dans un intervalle de temps raisonnable ? qu'est-ce d'ailleurs qu'un intervalle de temps raisonnable ?

• Étymologie

On ne peut pas parler d'*algorithme*, sans évoquer son étymologie qui renvoie à un grand scientifique du monde arabo-musulman. Le terme d'*algorithme* tire son origine du nom du mathématicien Al-Khwarizmi, né vers 780 dans la région du Kharezm (d'où son nom d'ailleurs), située au sud de la mer d'Aral, et mort vers 850. Il fut un des membres les plus importants de la Maison de la Sagesse à Bagdad, sorte d'académie où le calife Al-Mamun (dit le Sage) avait regroupé hommes et moyens pour le développement des sciences. Géographe, astronome, Al-Khwarizmi était aussi un brillant mathématicien.

Deux apports importants ont contribué à son rayonnement : tout d'abord, il est l'auteur du *premier* traité d'algèbre de toute l'histoire des mathématiques mais aussi d'un manuel d'arithmétique, traduit en latin et beaucoup utilisé dans l'Europe Médiévale, reprenant le calcul dit *indien*, c'est-à-dire les opérations permettant de calculer avec des nombres écrits dans la numération positionnelle de base 10 que nous connaissons aujourd'hui. Au Moyen-âge, le nom de cet auteur, déformé en *Algorismus*, désignait l'arithmétique et la science du calcul avec les nombres écrits en base 10.

• Exemples d'algorithmes

La notion d'algorithme est répandue, et en cherchant un peu, même dans la vie de tous les jours, les exemples ne manquent pas !

1) Une recette de cuisine, par exemple, peut être vue comme un algorithme : c'est bien une succession d'instruction permettant *en principe* d'obtenir un plat succulent.

2) Le dictionnaire des Sciences de Michel Serres et Nayla Farouki cite un exemple simple que tout un chacun rencontre chaque jour :

> *– Monsieur désire ?*
> *– Une baguette bien cuite.*
> *– Deux francs cinquante, s'il vous plaît.*
> *Vous tendez un billet de 50 francs et pendant que vous calculez, dans votre tête, 50 moins 2,5 égal 47,5, la vendeuse vous fait la conversation :*
> *– 50 centimes, ce qui fait trois, et 2 francs qui font 5, et 5, dix, et deux fois 20, cinquante. Le compte y est.*

Cette petite scène de la vie courante met en œuvre deux algorithmes : le vôtre qui est celui de la soustraction et celui de la boulangère qui arrive au même résultat par une méthode additive.

3) Pour multiplier deux entiers par exemple, la méthode que l'on apprend dès l'école primaire est un algorithme : c'est ce genre d'algorithme qu'Al-Khwarizmi a utilisé et transmis, on sait aujourd'hui avec quel succès !

4) L'extraction à la main de la racine carrée d'un entier est aussi un algorithme que l'on apprenait il n'y a pas si longtemps en classe de troisième, quand les calculatrices n'existaient pas. Il est de nos jours tombé en désuétude...

• Quelques points d'histoire

La notion d'algorithme, nous le signalions, est ancienne : de nombreux algorithmes étaient connus dès l'Antiquité dans le domaine de l'arithmétique ou de la géométrie. Citons pour mémoire les méthodes de résolution d'équations en nombres entiers, par Diophante au IV^{e} siècle de notre ère, ou encore le schéma de calcul du nombre π dû à Archimède.

Attardons-nous sur un des plus célèbres algorithmes de l'histoire des mathématiques, l'algorithme d'Euclide, permettant la détermination du PGCD de deux nombres entiers.
Pour le plaisir, citons la proposition I du livre VII des Éléments d'Euclide, celle qu'aujourd'hui on présente dans les livres d'arithmétique comme l'*algorithme* d'Euclide (pour Euclide, c'est une simple proposition et pas un algorithme !) :

> *Proposition I*
> *Deux nombres inégaux étant proposés, le plus petit étant toujours retranché du plus grand, si le reste ne mesure celui qui est avant lui que lorsque l'on a pris l'unité, les nombres proposés seront premiers entre eux.*

La proposition II du livre VII prouve que cette méthode des différences successives conduit en fait au PGCD des entiers initiaux.
On retrouve clairement dans ces deux propositions (vieilles de 23 siècles) tous les éléments qui servent aujourd'hui à caractériser un algorithme, à savoir :

1) des données (deux nombres *inégaux*) ;
2) un processus (des différences successives) s'exécutant étape après étape ;
3) une réponse clairement spécifiée, qui donne le PGCD des deux entiers dont on part ;
4) le nombre total d'étapes du processus est fini, quelles que soient les données.

Suivons pas à pas un exemple : on part donc de deux nombres et on remplace le plus grand par la différence des deux. Avec 42 et 17, on obtient successivement :

$$(42,17) \rightarrow (25,17) \rightarrow (8,17) \rightarrow (8,9) \rightarrow (8,1) \rightarrow (7,1) \rightarrow ... \rightarrow (1,1)$$

Les nombres sont égaux (donc le processus s'arrête) : ici le PGCD de 42 et de 17 est 1 (les nombres sont donc premiers entre eux).
Avec 45 et 36, on obtient :

$$(45,36) \rightarrow (9,36) \rightarrow (9,27) \rightarrow (9,18) \rightarrow (9,9)$$

Le processus s'arrête et le PGCD de 45 et 36 est 9.
Le processus ne comprend qu'un nombre fini d'étapes car on travaille avec des nombres entiers naturels, qui diminuent au fur et à mesure des calculs.
Pour terminer, signalons que la version moderne de cet algorithme est légèrement différente (mais équivalente) à celle d'Euclide : on procède par divisions successives.

Ces exemples anciens prouvent simplement que la notion d'algorithme est inhérente à l'activité mathématique. Avec les travaux des logiciens sur les fondements des mathématiques, elle devient dans les années 30 beaucoup plus qu'un simple outil permettant de résoudre tel ou tel problème mais un véritable *concept mathématique*. De plus, l'avènement des ordinateurs après la seconde guerre mondiale, a entraîné un renouvellement complet de l'algorithmique. Il est devenu possible depuis cette époque de traiter des problèmes de taille bien supérieure à celle des problèmes traités manuellement : pour preuve, citons les prodigieux progrès qui ont été fait après 1945 dans le calcul explicite des décimales de π (actuellement, on en connaît quelque 8 milliards) ou dans la recherche des grands nombres premiers. Des problèmes nouveaux sont aussi apparus, comme le tri ou la recherche rapide d'information dans une base de données.

L'efficacité des algorithmes

• Problèmes indécidables

La notion d'algorithme semble donc être la panacée des mathématiques... Dès qu'un problème se pose, sa résolution passe par la découverte d'un algorithme. Une question a alors intrigué les mathématiciens, question que nous avons déjà évoquée plus haut :

existe-t-il un algorithme pour n'importe quel problème donné ?

La réponse à cette question est non : dans les années 30, plusieurs mathématiciens (dont Kurt Gödel, Alonzo Church, Alan Turing et Emil Post) ont montré l'existence de problèmes qui ne peuvent pas être résolus par des algorithmes (c'est-à-dire pour lesquels il n'existe aucune méthode de résolution). Ces problèmes furent qualifiés d'*indécidables* (au sens *algorithmique* à ne pas confondre avec l'indécidabilité au sens logique de Gödel, dans le cadre d'une théorie axiomatique) et, en quelque sorte, ils marquent l'horizon de ce qu'un mathématicien peut traiter.

Les premiers problèmes indécidables rencontrés par les mathématiciens étaient souvent artificiels et très formels. Peu à peu, des problèmes simples (à énoncer !) de nature mathématique ou informatique furent *prouvés* indécidables. On peut citer le cas du dixième problème posé par David Hilbert lors du Congrès international de mathématiciens de Paris en 1900, dont l'énoncé est le suivant :

> *Étant donnée une équation diophantienne (c'est-à-dire à coefficients entiers), peut-on trouver un procédé qui détermine, par un nombre fini d'opérations, si cette équation possède des solutions en nombre entiers.*

Le problème est extrêmement général car il concerne n'importe quelle équation diophantienne et pas seulement une équation d'un type particulier. Remarquons que le mot algorithme est sous-jacent dans le discours de Hilbert, mais le concept lui-même n'est pas encore défini en 1900.

La réponse a été obtenue en 1970 par le mathématicien Yuri Matijasevic : il a montré que ce problème est en fait indécidable. En langage clair, il n'existe pas d'algorithme qui indique pour chaque équation diophantienne si elle a ou non des solutions en nombres entiers.

Déclarer un problème indécidable, c'est à la fois une grande victoire de l'esprit humain (une telle démonstration est souvent une prouesse mathématique) mais aussi, semble-t-il, un constat d'échec (c'est le glas de toute recherche sur ce problème). Ce dernier point est à relativiser car, si le problème dans sa généralité est définitivement clos, il peut cependant être résolu dans certains cas particuliers. Par exemple, Carl Ludwig Siegel, en 1972, a trouvé un algorithme pour les équations diophantiennes de degré 2 ; en revanche, on a aussi montré que le problème était indécidable pour le degré 4. Bref la recherche continue…

• Efficacité des algorithmes

Un algorithme doit-il finir dans un intervalle de temps raisonnable ? D'un point de vue théorique, avoir un nombre fini d'étapes est la seule contrainte que l'on impose à un algorithme ; mais d'un point de vue pratique, la réponse à un problème donné doit pouvoir être obtenue à échelle humaine. Échelle variable selon les besoins : quelques semaines parfois pour un calcul très lourd, quelques dixième de seconde tout au plus si la réponse donnée par l'algorithme permet au système de navigation d'un avion de corriger sa trajectoire.

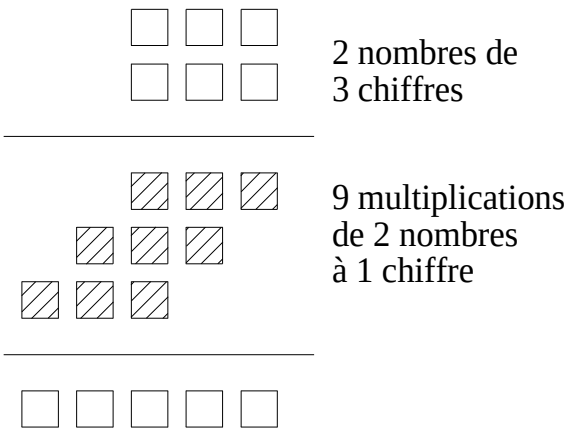
Ceci étant, l'existence théorique d'un algorithme ne suffit pas à ce que le problème soit concrètement résolu. Par exemple, l'algorithme pour factoriser le grand nombre RSA-210

RSA-210 =
24524664490027821197651766357308801846702678767833275974341445171506160083003858721695220839933207154910362682719167986407977672324300560059203563124656121846581790410013185929961993381701214933503487587055106 7
existe sans aucun doute. Il suffit par exemple de tester successivement si 2,3,5, etc. divisent un tel nombre. Au bout d'un nombre fini d'étapes, mais quelques dizaines de milliers d'années, une réponse sera donnée…

Les questions que nous abordons là concernent ce que l'on appelle l'*efficacité* des algorithmes : elles ont été étudiées vers le milieu des années 60 par A Cobham et J Edwards.

Pour évacuer le facteur temps, par trop subjectif, on fait intervenir le nombre d'étapes nécessaires pour mener l'algorithme à son terme quand il est écrit sur une *machine de Turing* (sorte d'ordinateur de référence). Dans les exemples qui suivent, je me contenterai de dénombrer les étapes mathématiques élémentaires (comme le nombre de multiplications ou de divisions) mais en toute rigueur, il faudrait faire ce travail pour une machine de Turing : sur le fond, cela ne change rien à la nature de l'algorithme du point de vue de son efficacité.

Un algorithme est dit *à complexité polynomiale* s'il existe deux entiers fixes A et k tels qu'à partir des données de longueur n, l'algorithme demande ***au plus*** An^k étapes.



Ainsi, la multiplication de deux entiers, suivant l'algorithme appris à l'école primaire, est à complexité polynomiale. Quand on multiplie deux entiers de n chiffres chacun (voir ci-contre), on effectue n^2 multiplications de deux entiers à un chiffre (sans tenir compte des retenues).
Signification concrète : s'il me faut 2 minutes pour multiplier deux entiers de 5 chiffres chacun (25 multiplications élémentaires), il m'en faudra quatre fois plus c'est-à-dire 8 minutes si je veux multiplier deux entiers de 10 chiffres chacun ($100 = 4 \times 25$ multiplications élémentaires)... On assiste là à une croissance implacable du temps demandé pour résoudre le problème posé mais cette croissance demeure raisonnable dans le cas d'une complexité polynomiale.

L'algorithme d'Euclide, dans sa version moderne (ou ancienne d'ailleurs), est aussi un algorithme polynomial (en fait meilleur qu'un algorithme strictement polynomial, mais quand même considéré comme un algorithme polynomial). Gabriel Lamé (1795-1870), ingénieur aux chemins de fer et, excusez du peu, considéré par Gauss comme le meilleur mathématicien français de son époque, a montré que le nombre de divisions nécessaires était inférieur à $5 \times \log n$, où n est le plus petit des deux nombres de départ. La faible croissance de la fonction logarithme explique la grande efficacité de l'algorithme d'Euclide : ainsi, si l'on veut déterminer le PGCD d'un entier de 100 chiffres et d'un autre de 150 chiffres, le calcul ne demandera qu'au plus $5 \times \log 100 = 10$ divisions, et donc un nombre total d'étapes très raisonnable.

Les algorithmes qui ne sont pas à complexité polynomiale sont dits *à complexité exponentielle*. Ainsi un algorithme qui requiert 2^n, n^n ou $n!$ étapes pour des données de longueur n est à complexité exponentielle : la fonction qui donne le nombre d'étapes n'est pas obligatoirement une fonction exponentielle au sens usuel.

Supposons pour fixer les idées qu'une étape fondamentale soit effectuée en 10^{-10} seconde : on peut alors confectionner le tableau suivant.

Complexité	Taille des données n				
	10	**20**	**50**	**80**	**120**
n	10^{-9} s	2×10^{-9} s	5×10^{-9} s	8×10^{-9} s	$1{,}2\times10^{-8}$ s
n^2	10^{-8} s	4×10^{-8} s	$2{,}5\times10^{-7}$ s	$6{,}4\times10^{-7}$ s	$1{,}44\times 10^{-6}$ s
n^3	10^{-7} s	8×10^{-7} s	$1{,}25\times10^{-5}$ s	0,000 05 s	0,000 1 s
$n!$	0,000 36 s	7,71 ans	$9{,}6 \times 10^{46}$ ans	$2{,}2 \times 10^{101}$ ans	$2{,}1 \times 10^{181}$ ans
2^n	10^{-7} s	10^{-4} s	31 h	38334 siècles	$4{,}2 \times 10^{18}$ ans

Dans ce tableau, on constate que les algorithmes qui demandent un temps raisonnable en fonction des données, sont ceux de complexité n, n^2 ou n^3 ; pour les autres, on assiste très vite à une véritable explosion de la durée en fonction de la taille des données, explosion qui interdit tout traitement pour des tailles de données pourtant petites. Ceci justifie la définition suivante.

Par définition, les algorithmes *efficaces* sont les algorithmes de complexité polynomiale ; les algorithmes *inefficaces* sont les algorithmes de complexité exponentielle.

La définition ne vaut que ce qu'elle vaut : c'est d'abord et avant tout une façon d'aborder le problème en classant les algorithmes. Cela étant, il faudra sûrement se méfier de l'*efficacité (au sens mathématique ci-dessus)* de certains algorithmes. Un algorithme de complexité $10^{100}n$ sera classé comme efficace ; pourtant même avec de petites valeurs de n, on attendra longtemps la réponse... À l'inverse un algorithme inefficace de complexité $10^{-100} \times 2^n$ fonctionnera sans problème pour des petites valeurs de n.

Cela semble limiter la portée de notre définition : en fait, les cas que l'on rencontre dans la pratique ne sont pas aussi caricaturaux. Par ailleurs, il est plus que probable que la notion d'efficacité d'un algorithme sera encore précisée à l'avenir.

• Classification des différents types de problèmes

L'approche précédente permet d'amorcer une classification des types de problèmes que peut rencontrer un mathématicien.

Tout d'abord, nous avons vu qu'il existait des *problèmes indécidables* (appelés aussi *problèmes sans preuve*), c'est-à-dire pour lesquels il n'existe pas d'algorithme : un exemple est le dixième problème de Hilbert.

Les autres problèmes nous intéressent car ce sont ceux pour lesquels existe un algorithme. Un problème est dit être *de type P* s'il existe pour ce problème un algorithme de résolution de complexité polynomiale (c'est le cas du problème du calcul du PGCD de deux entiers, résolu par l'algorithme d'Euclide).

À l'inverse, si **tous** les algorithmes de résolution d'un problème donné sont de type exponentiel, le problème est dit *à preuve difficile*. Les problèmes à preuve difficile ne sont pas pour autant une vue de l'esprit, comme on pourrait le croire de prime abord : c'est Michael Rabin, Juris Hartmanis et Richard Stearns qui ont démontré leur existence (toute théorique) dans les années 60. Par la suite, des problèmes plus « concrets » furent prouvés comme étant intrinsèquement difficiles. La situation est bien cernée, mais quelque peu désespérée, puisqu'on sait donc qu'*aucun* algorithme de type polynomial ne pourra résoudre un tel problème.

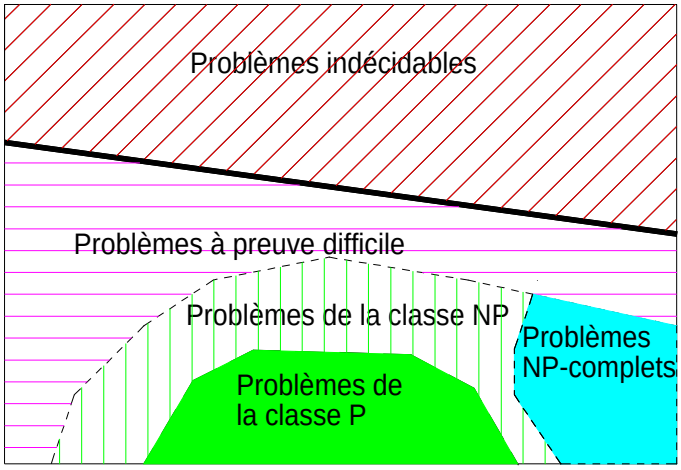
Entre les deux catégories précédentes se dessine une zone floue : ce n'est parce qu'on ne connaît que des algorithmes à complexité exponentielle pour résoudre un problème qu'on peut affirmer que le dit-problème est à preuve difficile.

On a donc défini une autre catégorie de problèmes, englobant la classe P, et chevauchant la zone floue décrite précédemment : c'est la *classe NP* (*non deterministic polynomial problem*)…

Comment décrire la classe NP ? Outre les problèmes de la classe P, elle regroupe des problèmes tous solubles (et donc pas indécidables) mais de type *incertain* : un algorithme *inefficace* de résolution existe mais l'on ne sait pas encore s'il est possible ou non de trouver un algorithme efficace pour ces problèmes. La factorisation d'un entier rentre dans cette catégorie.

Un dernier point important caractérise la classe NP : si j'ai beaucoup de chance, je peux **deviner** la solution d'un tel problème en un temps *polynomial*. Concrètement, cette solution est finalement assez simple à formuler bien qu'inaccessible aux moyens de calcul classique ; en langage courant, on sait « quelle tête elle

a », suffisamment en tout cas, pour être en mesure de la deviner (bien que cela demeure extrêmement peu probable). Ainsi, pour l'entier RSA-210 proposé plus haut, avec une chance insolente certes, mais on ne peut pas nier qu'il soit possible de deviner la solution... Ce côté *divinatoire* dans les mathématiques est troublant et ne semble pas faire bon ménage avec la rigueur.



Quel est donc l'intérêt de la classe NP ? Il réside en deux points.
Le premier point, c'est que de nombreux problèmes pour lesquels on n'a pas encore trouvé d'algorithme efficace se révèlent être de type NP.

Le second point, c'est que Stephen Cook a montré en 1971 qu'il était *hautement improbable* que l'on trouve pour certains problèmes NP (dits *NP-complets*) un algorithme efficace : en ce sens que si l'on trouve un algorithme efficace pour un tel problème, on en déduira un algorithme efficace pour *n'importe quel* problème NP. Or, croire que l'on résoudra le cas de *tous* les problèmes NP semble excessivement optimiste, donc on pense qu'on ne parviendra pas à résoudre le cas d'un problème NP-complet. Ceci étant, le problème demeure ouvert : c'est une question ardue que nous légueront sûrement aux mathématiciens du prochain siècle que je résume en

$$\mathbf{P \overset{?}{=} NP}$$

• Pour conclure

Pour conclure, signalons que, depuis les années 1990, on commence à parler d'ordinateurs *quantiques* : quelques essais concluants ont été réalisés ces derniers temps sur des machines-tests aux capacités dérisoires… L'on ne risque certes pas de voir se répandre ce genre de machine révolutionnaire avant quelques dizaines d'années. Les algorithmes tournant sur ces ordinateurs seraient complètement différents de ceux que l'on connaît aujourd'hui et les mathématiciens commencent à y réfléchir sérieusement : les résultats que l'on est susceptible d'obtenir sont particulièrement prometteurs.