Notes de cours
Semaine 10

Cours Turing

1 Trouver des grands nombres premiers

Comme nous l'avons vu la semaine dernière, il n'est a priori pas du tout aisé de vérifier qu'un nombre N à 100 chiffres est un nombre premier... Par exemple, vérifier que N n'est divisible par aucun nombre plus petit que lui (excepté 1 et lui-même) prend de l'ordre de 10^{100} divisions, ce qui est plus que le nombre estimé d'atomes dans l'univers !

On peut réduire sensiblement ce nombre d'opérations en observant la chose suivante : si un nombre N n'est pas premier et est donc le produit de deux nombres P et Q, c'est forcément le cas que soit P soit Q est plus petit ou égal à $\sqrt{N}$ (sinon, on obtiendrait que $P \cdot Q > \sqrt{N} \cdot \sqrt{N} = N$). Et donc, pour tester si un nombre est premier, il suffit de tester tous ses diviseurs potentiels de 2 jusqu'à $\sqrt{N}$, et non de 2 jusqu'à N. Alors certes, $\sqrt{N}$ est beaucoup plus petit que N, mais lorsque $N = 10^{100}$, on obtient que $\sqrt{N} = 10^{100/2} = 10^{50}$, ce qui représente un nombre encore extrêmement grand !

Il existe heureusement une autre façon beaucoup plus efficace de procéder ! Mais elle demande un peu de travail...

1.1 Arithmétique modulaire

Commençons par une définition : étant donné un nombre entier positif N, on dit que "A modulo N est égal à B" et on écrit :

$$A \,(\text{mod } N) = B$$

si B est le reste de la division de A par N. B est donc un nombre compris entre 0 et $N - 1$. Lorsqu'on effectue des opérations modulo N, on peut se limiter à ces nombres, ce qui, comme nous allons le voir, n'est pas sans intérêt !

Voici un exemple avec $N = 11$ et $A = 64$, on a $A = 5 \cdot 11 + 9$ donc $A \,(\text{mod } 11) = 9$.

Et voici comment effectuer des additions et multiplications modulo N (toujours avec $N = 11$) :

- Si $A = 64$ et $B = 17$, alors $(A + B) \,(\text{mod } 11) = 81 \,(\text{mod } 11) = 4$. Mais notez qu'on aurait pu aussi calculer d'abord :

$A \,(\text{mod } 11) = 64 \,(\text{mod } 11) = 9 \quad$ et $\quad B \,(\text{mod } 11) = 17 \,(\text{mod } 11) = 6$

et effectuer ensuite $(9 + 6) \,(\text{mod } 11) = 15 \,(\text{mod } 11) = 4$ pour arriver au même résultat.

- Toujours avec $A = 64$ et $B = 17$, on peut effectuer directement $(A{\cdot}B) \,(\text{mod } 11) = 1088 \,(\text{mod } 11) = 10$, mais on peut aussi se simplifier la vie en utilisant à nouveau le fait que $A \,(\text{mod } 11) = 9$, $B \,(\text{mod } 11) = 6$, puis en calculant $(9 \cdot 6) \,(\text{mod } 11) = 54 \,(\text{mod } 11) = 10$.

Nous avons utilisé ici le fait que les opérations d'addition et de multiplication commutent chacune avec l'opération modulo (ce qui signifie qu'on peut effectuer ces opérations dans l'ordre qu'on désire). C'est très pratique, car cela permet de ne jamais faire de calculs impliquant des nombres plus grands que N.

1.2 Le petit théorème de Fermat

Maintenant que nous avons vu quelques opérations de base en arithmétique modulaire, en quoi donc cela peut-il nous être utile pour trouver des nombres premiers ? La première relation avec les nombres premiers est la suivante, aussi connue sous le nom de *petit théorème de Fermat*[1] :

Si N est un nombre premier, alors $A^{N-1} \,(\text{mod } N) = 1$ pour tout nombre entier $1 \leq A < N$.

Malheureusement, l'assertion ne va pas dans le bon sens ! En effet, si on trouve un nombre $1 \leq A < N$ tel que $A^{N-1} \,(\text{mod } N) = 1$, ceci n'implique pas a priori que N est un nombre premier. Par contre, on peut utiliser la *contraposée* de ce théorème :

S'il existe un nombre entier $2 \leq A < N$ tel que $A^{N-1} \,(\text{mod } N) \neq 1$, alors N n'est *pas* un nombre premier (noter qu'il est impossible ici que $A = 1$, car $1^{N-1} \,(\text{mod } N) = 1$ pour tout N).

Ainsi, on a un outil pour dénicher des grands nombres qui ne sont *pas* premiers. "A quoi bon ?" direz-vous, car ce qui nous intéresse, ce sont les grands nombres premiers... Il se trouve qu'une extension du petit théorème de Fermat va nous aider. Celle-ci dit la chose suivante :

S'il existe un nombre $2 \leq A \leq N - 1$ tel que $\text{PGDC}(A, N) = 1$ et $A^{N-1} \,(\text{mod } N) \neq 1$, alors $B^{N-1} \,(\text{mod } N) \neq 1$ pour au moins la moitié des autres nombres B compris entre 2 et $N - 1$.

Ce que dit ce résultat pour l'essentiel (si on fait abstraction d'un petit détail ; voir la 2$^{\text{e}}$ remarque ci-dessous), c'est : *si on choisit un nombre A uniformément au hasard entre 2 et $N-1$ et qu'on trouve que $A^{N-1} \,(mod\ N) = 1$, alors il y a au moins 50% de chances pour que N soit un nombre premier.* En effet, puisque si N n'était pas premier, au moins la moitié des nombres A entre 2 et $N-1$ donneraient $A^{N-1} \,(\text{mod } N) \neq 1$). Nous allons maintenant voir en détail quelle utilisation faire de ce résultat pour transformer l'essai, car il est clair qu'on ne peut pas se satisfaire d'un algorithme qui se trompe une fois sur deux !

1. Attention à ne pas confondre ici avec le *grand* ou *dernier* théorème de Fermat, qui est resté un problème ouvert pendant de nombreuses années, plus précisément de 1637, date de son énoncé, à 1995, date de la parution de sa démonstration par Andrew Wiles.

1.3 Bis repetita...

L'idée est maintenant de répéter l'expérience précédente, en tirant plusieurs nombres $A_1, A_2, \ldots, A_k$ au hasard, chacun entre 2 et $N-1$, et indépendamment les uns des autres (d'où l'importance d'avoir un bon générateur de nombres aléatoires). Pour tous ces nombres, on teste si

$$A_1^{N-1} \pmod N = 1, \quad A_2^{N-1} \pmod N = 1, \quad \ldots, \quad A_k^{N-1} \pmod N = 1$$

- Si on n'obtient pas le résultat 1 pour un des nombres $A_1, A_2, \ldots, A_k$, alors on sait par le petit théorème de Fermat (plus précisément, par sa contraposée), que N n'est pas premier. Fin de la discussion : il faut essayer une nouvelle valeur de N.

- Si par contre c'est le cas qu'on obtient le résultat 1 pour chacun des nombres $A_1, A_2, \ldots, A_k$, alors qu'est-ce que ça nous dit ? Par ce qui précède, *si N n'est pas un nombre premier*, alors il y a au moins 50% de chances, pour chaque nombre A, que $A^{N-1} \pmod N \neq 1$. Donc dans ce cas, il faudrait être vraiment malchanceux pour obtenir le résultat 1 pour chacun des nombres $A_1, A_2, \ldots, A_k$. Plus précisément, la chance que ça arrive vaut au plus :

$$p = \underbrace{\frac{1}{2} \cdot \frac{1}{2} \cdots \frac{1}{2}}_{k \text{ fois}} = \frac{1}{2^k}$$

Pour une valeur même assez petite de k, cette probabilité p est très proche de 0. Par exemple : si $k = 10$, alors $p \simeq 0,001$; si $k = 20$, alors $p \simeq 0,000001$, et si $k = 30$, alors $p \simeq 0,000000001$. En conclusion : si on n'obtient que des résultats 1 aux k tests ci-dessus, on peut déclarer avec confiance que N est un nombre premier (à noter que si N est un nombre premier, alors par le petit théorème de Fermat, on sait qu'on obtiendra toujours le résultat 1).

Remarques

- Il peut sembler assez troublant que pour vérifier une propriété déterministe d'un nombre entier, à savoir ici sa *primalité*, on ait recours au hasard. Par ce qui précède, vous voyez cependant que ce hasard peut être réduit "autant qu'on veut", en assez peu d'étapes. En pratique, c'est tout à fait acceptable ! Il existe d'ailleurs une pléthore d'algorithmes qui font appel au hasard pour résoudre des problèmes déterministes. Sans le hasard, nos ordinateurs modernes seraient beaucoup moins puissants !

- Un détail a été omis dans la présentation qui précède : il existe quelques nombres entiers N (pas si nombreux, mais quand-même), qui possèdent la propriété étrange de ne pas être premiers tout en vérifiant $A^{N-1} \pmod N = 1$ pour *toutes* les valeurs de A entre 2 et $N-1$. Ces nombres sont appelés les *nombres de Carmichael*. Le plus petit d'entre eux est $N = 561 = 17 \cdot 33$. Pour gérer ce problème, un autre algorithme est nécessaire : l'*algorithme de Miller-Rabin*.

Et pour finir, reste une question cruciale : est-ce qu'avec tout ce travail, nous avons gagné quoi que ce soit par rapport à l'algorithme qui teste tous les diviseurs de N allant de 2 à $\sqrt{N}$? ? ? Faisons un peu les comptes pour voir...

Comme mentionné au début de ce chapitre, l'algorithme classique demande d'effectuer de l'ordre de 10^{50} divisions pour vérifier qu'un nombre à 100 chiffres est premier. De plus, comme déjà vu la semaine dernière, en tirant au hasard un nombre à 100 chiffres, on a environ une chance sur 230 de tirer un nombre premier. Au total, on trouvera donc avec cette méthode un nombre premier après

$$230 \cdot 10^{50} \text{ opérations}$$

en moyenne ; autrement dit, mieux vaut être patient !

Comparons maintenant avec la méthode vue plus haut : certes, on n'échappe pas au fait qu'il faut tirer de l'ordre de 230 nombres N au hasard pour finalement tomber sur un nombre premier. Mais combien d'opérations coûte à chaque fois le test proposé ci-dessus ? Pour fixer les idées, disons qu'on effectue le test avec $k = 30$ nombres A tirés au hasard (ce qui rappelons-le mène à une probabilité d'erreur inférieure ou égale à 0,000000001) : pour chaque nombre, ceci consiste à calculer

$$A^{N-1} \pmod N \tag{1}$$

Heureusement, comme nous allons le voir ci-dessous, pour un nombre N à 100 chiffres, ce calcul ne demande pas plus de deux millions d'opérations. Donc au total, le nombre d'opérations à effectuer pour trouver un nombre premier à 100 chiffres avec cette méthode est de l'ordre de

$$230 \cdot 30 \cdot 2 \text{ millions} \simeq 10 \text{ milliards}$$

Or 10 milliards d'opérations, avec un ordinateur moderne, ça se fait très vite (en tout cas beaucoup plus vite que $230 \cdot 10^{50}$ opérations !).

Reste à comprendre pourquoi l'opération (1) ne demande pas plus de deux millions d'opérations. Pour cela, il nous faut (clairement...) revenir un moment à l'arithmétique modulaire.

1.4 Exponentiation rapide ("square-and-multiply")

Nous avons vu au début de ce chapitre que "prendre des modulos" permet de simplifier de additions et multiplications, mais c'est encore plus vrai lorsqu'on effectue une opération d'*exponentiation* (qui commute également avec l'opération modulo) : pour calculer $A^B \pmod N$, on peut bien sûr d'abord calculer A^B, puis prendre la reste de la division de ce nombre par N pour trouver le résultat, mais il est clairement plus facile d'utiliser le fait que

$$A^B \pmod N = (A \pmod N)^B \pmod N$$

Voyons ça sur un exemple avec $N = 11$, $A = 64$ et $B = 6$: pour calculer $64^6 \pmod{11}$, calculons d'abord $64 \pmod{11} = 9$, puis

$$64^6 \pmod{11} = 9^6 \pmod{11} = 531'441 \pmod{11} = 9$$

??? A vous entendre, vous ne semblez pas forcément convaincus... Certes, il est plus facile de calculer 9^6 que 64^6, mais ça reste quand-même un calcul ardu ! En fait, on peut faire mieux que ça. Regardez plutôt :

$$9^6 = 9^{4+2} = 9^4 \cdot 9^2 = (9^2)^2 \cdot 9^2 \quad \text{donc} \quad 9^6 \pmod{11} = (9^2 \pmod{11})^2 \pmod{11} \cdot (9^2 \pmod{11})$$

Pour effectuer ce calcul, on calcule d'abord $9^2 \pmod{11} = 81 \pmod{11} = 4$, puis $4^2 \pmod{11} = 16 \pmod{11} = 5$, et finalement la multiplication $(5 \cdot 4) \pmod{11} = 20 \pmod{11} = 9$, qui nous donne le résultat voulu : $64^6 \pmod{11} = 9$, sans avoir eu à effectuer de multiplications plus compliquées que des livrets appris à l'école primaire.

Remarque : Attention ! Pour calculer $A^B \pmod{N}$, on peut remplacer A par $A \pmod{N}$, mais on ne peut *pas* remplacer B par $B \pmod{N}$. Par exemple :

$$64^{17} \pmod{11} \neq 64^6 \pmod{11}$$

En général, voyons maintenant comment cette méthode fonctionne : pour calculer A^B tout d'abord (en oubliant mod N pour l'instant), on utilise la décomposition binaire de B, c'est-à-dire qu'on écrit B comme une somme de puissances de 2. Par exemple, pour $B = 43$, écrivons $B = 43 = 32 + 8 + 2 + 1$, et donc

$$A^B = A^{32+8+2+1} = A^{32} \cdot A^8 \cdot A^2 \cdot A$$

Ainsi, en calculant successivement A, A^2, $A^4 = (A^2)^2$, $A^8 = (A^4)^2$, $A^{16} = (A^8)^2$ et $A^{32} = (A^{16})^2$, on n'effectue que des carrés, et pour obtenir le résultat A^B, il suffit de multiplier les quatre termes A, A^2, A^8 et A^{32}. Cette méthode, qui s'appelle en anglais "square-and-multiply", permet d'éviter le calcul direct de A^{43}, qui est fastidieux.

Viennent maintenant s'ajouter les modulos : comme déjà mentionné ci-dessus, l'avantage d'effectuer des opérations modulo permet de se cantonner aux nombres compris entre 0 et $N - 1$. Ainsi, si A^B est potentiellement un (très) grand nombre, $A^B \pmod{N}$ sera par contre toujours un nombre compris entre 0 et $N - 1$. Donc on peut refaire tout ce qu'on vient de faire en prenant chaque fois systématiquement le résultat modulo N. Ainsi, on sait qu'on n'effectuera que des carrés ou des multiplications avec des nombres compris entre 0 et $N - 1$.

Comptons maintenant le nombre d'opérations effectuées, en supposant que A, B, N sont tous des nombres à 100 chiffres :

- effectuer le carré d'un nombre à 100 chiffres ou la multiplication de deux nombres à 100 chiffres coûte $100 \cdot 100 = 10'000$ opérations environ (penser à la multiplication en colonnes) ;

- si B est un nombre à 100 chiffres, sa decomposition binaire sera également composée d'une centaine de termes (à peu près) ;

- et donc pour calculer la série A, A^2, A^4, A^8... jusqu'à $A^{2^{100}}$ (le tout modulo N), il faudra effectuer environ 100 carrés, et encore 100 autres multiplications (au pire) des termes dont on a besoin pour obtenir le résultat final A^B. Ceci représente au total

$$2 \cdot 100 \cdot 10'000 = 2 \text{ millions d'opérations}$$

comme annoncé plus haut ! Nous voilà donc arrivés à notre but, à savoir : effectuer un nombre raisonnable d'opérations pour trouver un (très) grand nombre premier.

2 Factoriser des grands nombres entiers

Maintenant que nous avons trouvé un algorithme efficace pour trouver des grands nombres premiers, il semble logique de passer à la prochaine étape et essayer de trouver un algorithme efficace pour exprimer un grand nombre en produit de facteurs premiers. En effet, c'est un fait établi qui que tout nombre entier peut toujours s'écrire comme un produit de facteurs premiers. Voici quelques exemples :

$98 = 2 \cdot 7 \cdot 7$

$99 = 3 \cdot 3 \cdot 11$

$100 = 2 \cdot 2 \cdot 5 \cdot 5$

$101 = 101$

$102 = 2 \cdot 3 \cdot 17$

etc.

Pour autant, existe-t-il un algorithme efficace permettant de calculer la factorisation d'un grand nombre N, par exemple un nombre à 100 chiffres ? La réponse honnête à cette question est : "A l'heure actuelle, on ne sait pas" ! Mais une chose est sûre : jusqu'à présent, on n'a pas trouvé d'algorithme efficace [2].

Donc pour aussi étonnant que cela puisse paraître, nous avons ici affaire à deux problèmes de même nature (le problème de la primalité et celui de la factorisation), qui sont en fait complètement différents, puisqu'une solution a été trouvée pour le premier, tandis que le second résiste encore... Cette différence fondamentale entre ces deux problèmes est même utilisée comme principe de base du système de cryptographie à clé publique RSA, que nous verrons dans les prochaines leçons.

2. Il existe en fait un algorithme efficace : l'algorithme de Shor, mais celui-ci requiert l'utilisation d'un ordinateur quantique... pas encore construit !