

Factorisation des grands nombres

Decker Benjamin - Le Boulc'h Erin

Double Licence Mathématiques Informatique - Semestre 6

Sommaire

1	Introduction	2
2	Factorisation naïve par divisions successives	2
3	Méthode de Fermat	5
4	Méthode de Kraitchik	7
5	Références	10

1 Introduction

D'après le théorème fondamental de l'arithmétique, tout entier supérieur à 1 admet une unique représentation comme produit de nombres premiers. En revanche, une telle décomposition peut être très difficile à déterminer. C'est justement sur ce principe qu'a été créé le chiffrement RSA à la fin des années 70, aujourd'hui largement utilisé pour sécuriser la plupart des informations en informatique. En effet, cette méthode tire partie du fait qu'il est très simple de multiplier deux nombres mais qu'il peut être très long d'extraire les facteurs premiers d'un nombre composé. Ainsi en 1976, on estimait qu'avec les algorithmes de l'époque il aurait fallu plus de 10^{20} années pour factoriser $2^{251} - 1$, un nombre à 76 chiffres. De nos jours, il est possible d'obtenir ce résultat avec un ordinateur de bureau en moins d'une minute. L'enjeu de la factorisation des grands nombres est donc de créer des algorithmes capables de factoriser des nombres arbitrairement grands en des temps raisonnables, dans le but de "casser" le code RSA. Nous allons donc voir quelques méthodes, mises au point à différentes époques, permettant de factoriser des nombres de manière plus ou moins efficace.

Objectif : Explorer les méthodes de factorisation, notamment pour de grands entiers

2 Factorisation naïve par divisions successives

Principe : trouver le plus petit diviseur de n , i.e diviser n par tous les nombres premiers plus petits que lui et voir si un d'eux le divise, puis itérer la méthode sur le quotient, jusqu'à arriver à 1. On récupère les diviseurs et on obtient la factorisation de n .

Exemple Soit $n = 28$.

$$28 \div 2 = 14$$

$$14 \div 2 = 7$$

$$7 \div 7 = 1$$

On obtient donc $28 = 2 \times 2 \times 7$

Lemme Le plus petit diviseur de N ne dépasse pas \sqrt{N} .

En effet, prenons p le plus petit diviseur de N . Alors $N = p \times q$ avec q un entier ($p \leq q$ par minimalité de p). On a $p \leq p \times q = N$ donc $p \leq \sqrt{N}$.

Critère de finitude Soit N l'entier que l'on veut factoriser.

Si N est premier, alors, l'algorithme se finit quand on a testé tous les nombres premiers inférieurs à N : aucun diviseur n'ayant été trouvé, c'est comme cela qu'on en déduit qu'il est premier, i.e qu'il n'a pas de diviseur premier autre que 1 et lui-même.

Si N n'est pas premier, alors il est divisible par plusieurs entiers premiers plus petits que lui. La méthode permet de les trouver dans l'ordre croissant et si on considère les quotients successifs comme une suite, celle-ci est strictement décroissante et minorée par 1 donc elle est finie.

Optimisation

- On peut effectuer l'algorithme sur les $p - 1$ diviseurs plus petits de N (avec p le nombre de facteurs premiers dans la factorisation de N) et ajouter le dernier quotient (qui est un entier premier) ce qui permet de ne tester que les entiers qui ne dépassent pas la racine du quotient à chaque itération.

Si $N = p_1 \times \dots \times p_q$ avec $p_1 \leq \dots \leq p_q$ premiers, pour $i \in \{1, \dots, q-1\}$, p_i est le plus petit diviseur premier de $l = p_i \times \dots \times p_q$ donc p_i ne dépasse pas \sqrt{l} .

- Si $d|N$, il faut ajouter d aux diviseurs et continuer l'algorithme sur le quotient de la division. Si $d \nmid N$, on sait que tous ses multiples ne divisent pas N non plus. Si on teste chaque diviseur, le nombre de division est linéaire. Ainsi, pour de très grands nombres, il pourrait être intéressant de ne pas faire de divisions inutiles. Ici, si on a une liste des entiers premiers à tester, on pourrait donc retirer ces multiples. Pour avoir une réelle optimisation, il faudrait faire attention à la suppression de ces valeurs de la liste. En effet, si on ne fait pas cette opération avec une complexité en temps constant, on ne gagne rien, voire on fait plus d'opérations que si on laissait les divisions inutiles.

On se rend en même temps compte que pour des nombres très grands, stocker \sqrt{N} entiers de plus en plus grands n'est pas viable. Il faut donc trouver une alternative.

- Si on utilise une variable qu'on incrémente à chaque fois au lieu de stocker les entiers dans une liste, on se retrouve dans le cas où on fait des divisions inutiles (potentiellement BEAUCOUP de divisions inutiles).

Quand on fait cette méthode sur de petits entiers comme dans l'exemple, c'est assez facile car on connaît par coeur les petits nombres premiers ou on peut y avoir facilement accès, or la question suivante se pose pour les entiers très (très) grands : comment savoir par quels entiers diviser pour espérer trouver un diviseur premier ? Faut-il diviser par tous les entiers (pas forcément premiers) plus petits que \sqrt{N} , en utilisant un compteur par exemple, au risque d'avoir un très grand nombre de divisions à effectuer (il y a beaucoup plus de nombres entiers que de nombres premiers) ? Faut-il stocker au préalable dans une liste les nombres entiers inférieurs à \sqrt{N} pour réduire le nombre de divisions nécessaires ? Et dans ce cas là, comment établir cette liste (Crible d'Erathostène, application des divisions successives à chaque entier intermédiaire) avec une complexité acceptable dans tous les cas ? (Dans un cas extrême, pour une puissance de 2 très grande, cette méthode serait inefficace car il suffirait de diviser successivement par 2, au lieu de construire cette liste).

Conclusion On observe que sur le principe, utiliser les divisions successives pour établir une factorisation naïve fonctionne toujours (avec le critère de finitude). Cependant, cette méthode est limitée en application pour de très grands entiers, soit en termes d'opérations, soit en termes d'espace et il n'est pas possible de trouver un compromis efficace entre les deux complexités.

Implémentation Nous avons implémenté cette méthode en JAVA. La représentation des entiers dans ce langage de programmation nous permet de tester notre algorithme de manière certaine sur des entiers jusqu'à 10 000. Au delà, les nombres ayant une factorisation simple sont factorisables (comme $25590 = 2 \times 3 \times 5 \times 853$) mais d'autres sont indiqués comme des nombres premiers alors que ce n'est pas le cas (comme $25591 = 157 \times 163$). Il est possible d'obtenir de meilleurs résultats avec d'autres langages de programmation par exemple.

3 Méthode de Fermat

Lemmes nécessaires

1. Tous les nombres premiers supérieurs à 2 sont impairs.
2. Une somme ou une différence de nombres impairs est paire.
3. Un entier se décompose comme $2^k N$ avec $k \in \mathbb{N}$, N un entier impair.
Si N est premier, on a donc la décomposition en facteurs premiers, sinon, N admet une factorisation non triviale.
4. Il existe une bijection entre $\{(a, b) \in \mathbb{N}^2, a \leq b, N = ab\}$ et $\{(r, s) \in \mathbb{N}^2, N = r^2 - s^2\}$ (On peut poser $(r = \frac{a+b}{2}, s = \frac{a-b}{2})$ et $(a = r + S, b = r - s)$.)

Comme on souhaite $a, b \in \mathbb{N}^2$ tel que $N = ab$ et que toute la difficulté de la factorisation consiste à trouver ces valeurs, la Méthode de Fermat utilise donc la bijection du lemme 4. pour trouver plus facilement des diviseurs de N à l'aide de différences de carrés. Elle est particulièrement efficace quand N est le produit d'entiers proches.

Méthode On sait qu'un entier est décomposable comme $2^k N$, avec N un entier impair. Si N est premier, on a déjà la décomposition en facteurs premiers qui nous intéresse. Sinon, nous allons décomposer N en produit de deux facteurs impairs (auxquels on appliquera la même procédure s'ils ne sont pas premiers).

On veut trouver R et S tel que $N = R^2 - S^2 = (R+S)(R-S)$. Pour cela, on commence par poser $c = \lfloor \sqrt{N} \rfloor$ et on va y ajouter $k \in \mathbb{N}$ de plus en plus grand, jusqu'à trouver un S^2 convenable. En effet, on pose $R = c + k$ (R est forcément plus grand que \sqrt{N}) dès que k est non nul) pour avoir $S^2 = N - R^2 = N - (c + k)^2$. Un S^2 convenable est donc tel que $S \in \mathbb{N}$. Une fois R et S trouvés, on pourra donc écrire $N = (R + S)(R - S)$.

Exemple Factorisons 15 par la méthode de Fermat.

(On sait que $15 = 3 \times 5$)

15 étant impair, on a $15 = 2^0 N$ avec $N = 15$.

$c = \lfloor \sqrt{15} \rfloor = 3$

Si on prend $R = c + 0 = 3$, alors $R^2 = 9$ et $S^2 = N - R^2 = 15 - 9 = 6$.

Or $\sqrt{6}$ n'est pas un entier.

On continue avec $R = c + 1 = 4$.

On a $R^2 = 16$ et $S^2 = N - R^2 = 15 - 16 = -1$.

On a donc $S = 1$ qui est $\sqrt{1}$

On retrouve bien $N = (R + S)(R - S) = (4 + 1) \times (4 - 1) = 5 \times 3$

Limites Cette méthode est rapide pour factoriser des nombres qui sont le produit de deux entiers proches. Pour factoriser 2041 par exemple, il faut aller jusqu'à $k = 40$. Avec la méthode suivante, le nombre d'opérations est moins important.

Implémentation Nous avons implémenté cette méthode en JAVA et en OCAML.

4 Méthode de Kraitchik

Principe Pour factoriser un entier N impair, nous allons utiliser le principe suivant : on identifie u et v des entiers tels que :

- $N \mid (u+v)(u-v) \iff u^2 \equiv v^2[N]$
- $N \nmid (u+v)$ et $N \nmid (u-v) \iff u \not\equiv \pm v[N]$

On obtient ainsi :

$$N = \text{pgcd}(u+v, N) \times \text{pgcd}(u-v, N)$$

Pour trouver de tels u et v , nous considérons le polynôme de Kraitchik : $Q(X) = X^2 - N \in \mathbb{Z}[X]$.

(*) Prenons une famille d'entiers $(x_i)_{i \in I}$ tel que $\forall i \in I, \prod_{i \in I} Q(x_i) = v^2, v \in \mathbb{N}$. On pose $\prod_{i \in I} x_i^2 = u^2$.

Alors on a :

$$\prod_{i \in I} x_i^2 \equiv \prod_{i \in I} Q(x_i)[N]$$

On admet qu'il y a une grande possibilité que $u \not\equiv v[N]$. Il suffit donc de trouver une telle famille $(x_i)_{i \in I}$.

Exemple Factorisons $N = 2041$ avec cette méthode.

Le produit des polynômes de Kraitchik doit être un carré donc positif. Ainsi, on regarde les carrés plus grands que 2041 pour constituer notre famille. Comme $\lfloor \sqrt{2041} \rfloor = 45$, regardons

$$x_1 = 46, x_2 = 47, x_3 = 48$$

$$x_4 = 49, x_5 = 50, x_6 = 51$$

On a

$$Q(x_1) = 3 \times 5^2, Q(x_2) = 2^3 \times 3 \times 7, Q(x_3) = 263,$$

$$Q(x_4) = 2^3 \times 3^2 \times 5, Q(x_5) = 3^3 \times 17, Q(x_6) = 2^4 \times 5 \times 7$$

On remarque que $Q(x_1) \times Q(x_2) \times Q(x_4) \times Q(x_6)$ est un carré (toutes les puissances de nombres premiers sont paires). On pose donc

$$u = 46 \times 47 \times 49 \times 51$$

$$v = 2^5 \times 3^2 \times 5^2 \times 7$$

En calculant $\text{pgcd}(u+v, N) = 157$ et $\text{pgcd}(u-v, N) = 13$, on obtient donc $N = 13 \times 157$.

Trouver ce qui nous intéresse Nous allons montrer qu'il est toujours possible de trouver une famille $(x_i)_{i \in I}$ telle que $\prod_{i \in I} Q(x_i)$ soit un carré grâce à l'algèbre linéaire.

Lemme Soit $k, B \in \mathbb{N}$ tel que $k \geq \pi(B) + 1$, (m_1, \dots, m_k) des entiers naturels B-friables, il existe $I \subset \{1, \dots, k\}$ tel que $\prod_{i \in I} m_i$ est un carré.

Démonstration Soit $B \in \mathbb{N}$, $x_i \in \mathbb{N}$, $x_i \geq \lfloor \sqrt{N} \rfloor + 1$. Fixons

$$r \in \mathbb{N}, r = \pi(B)$$

p_1, \dots, p_r les nombres premiers $\leq B$

$a_1, \dots, a_m, (m > r)$ B-friables

$\forall i \in \{1, \dots, k\}$, on note $a_i = \prod_{j=1}^r p_j^{\alpha_{i,j}}$, $\alpha_{i,j} \geq 0$ (par hypothèse, tous les a_i peuvent bien s'écrire sous cette forme puisqu'ils sont B-friables.)

On observe que

$$\begin{aligned} \prod_{i=1}^m a_i \text{ est un carré} &\iff \prod_{i=1}^m \prod_{j=1}^r p_j^{\alpha_{i,j}} \text{ est un carré} \\ &\iff \prod_{j=1}^r p_j^{\sum_{i=1}^m \alpha_{i,j}} \text{ est un carré} \\ &\iff \sum_{i=1}^m \alpha_{i,j} \text{ est pair } \forall j \in \{1, \dots, m\} \end{aligned}$$

On pose M la matrice dans \mathbb{F}_2 de taille (m, r) avec aux coordonnées (i, j) , $\alpha_{i,j}$ modulo 2. Comme $rg(M) \leq r < m$, les vecteurs lignes forment un système lié, donc il existe $I \subset \{1, \dots, m\}$ tel que $\sum_{i \in I} l_i = 0$ et donc il existe une famille $(a_i)_{i \in I}$ telle que $\prod_{i \in I} a_i$ est un carré.

Si on pose $\lambda_i = 1 \iff a_i \in I$, chaque vecteur de \mathbb{F}_2^m du noyau de l'application linéaire dont la matrice est la transposée de M permet d'obtenir une famille $(a_i)_{i \in I}$ telle que $\prod_{i \in I} a_i$ est un carré. Notons que le noyau est non nul par le théorème du rang.

Ainsi, on a montré qu'en prenant suffisamment d'éléments a_i B-friables, c'est à dire pour que la matrice M ne soit pas carrée, il existe forcément une famille $(a_i)_{i \in I}$ telle que $\prod_{i \in I} a_i$ est un carré.

En pratique Pour factoriser N (supposant qu'il ne s'agit pas d'une puissance d'un nombre premier), on prend donc :

$$(x_i)_{i \in I} \text{ avec } \forall i \in I, x_i \in \mathbb{N}, x_i > \lfloor \sqrt{N} \rfloor$$

$$(a_i)_{i \in I}, \forall i \in I a_i = Q(x_i) = x_i^2 - N$$

Exemple Reprenons l'exemple précédent avec $N = 2041$.

Choix de B

Implémentation Nous avons implémenté cette méthode en JAVA.

5 Références

Nous nous sommes largement appuyés sur CHAPITRE VI - MÉTHODES DE FACTORISATION du cours de Cryptographie d'ALAIN KRAUS en 2009-2010 à l'Université Pierre et Marie Curie. Nous avons également pu avoir des précisions, notamment sur l'aspect algébrique des méthodes présentées, de la part de notre encadrant de projet OLIVIER BRUNAT, enseignant chercheur de l'IMJ-PRG à l'Université Paris Cité.

Voici une liste de ressources que nous avons consulté dans une moindre mesure :

- Démonstrateur du crible quadratique - Université de Technologie de Compiègne
- La factorisation des grands entiers : de Fermat au code RSA - tangente-mag.com
- Méthode de factorisation de Fermat - Wikipedia