DM: Test de primalité de Solovay-Strassen

leo.perrin@inria.fr

Pour le 22 avril 2019

Les tests de primalité sont des algorithmes indispensables pour la cryptographie à clé publique. Parmi leurs utilisations les plus courantes il y a la phase de génération des clés pour le cryptosystème RSA ou les échanges de clés basés sur le protocole de Diffie-Hellman.

Les tests de primalité les plus efficaces sont les tests de nature probabiliste. Ces tests affirment qu'un nombre est composé avec une certitude absolue, mais ils peuvent se tromper en déclarant qu'un nombre est premier. Pour diminuer la probabilité d'erreur, le test est répété plusieurs fois (avec des paramètres à chaque fois différents). Si après plusieurs itérations, le nombre testé n'a pas été reconnu comme composé, on conclut alors qu'il est probablement premier.

Le but de ce DM est d'implémenter en `C` un test de primalité très connu : le test de Solovay-Strassen. Ce test a été développé par Robert Solovay et Volker Strassen en 1977. Il emploie ce qu'on appelle le *symbole de Jacobi* qui est lui-même une généralisation du *symbole de Legendre*.

1 Prérequis mathématiques

1.1 Le symbole de Legendre

Soit p un nombre premier impair. On dit qu'un entier a est un **résidu quadratique modulo** p, si $a \mod p$ est un carré dans $\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$, c'est-à-dire s'il existe un entier b tel que $a \equiv b^2 \mod p$. Dans le cas contraire on dit que a est un **non-résidu quadratique modulo** p.

On définit alors le **symbole de Legendre** $\left(\frac{a}{p}\right)$ comme suit :

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{si } p \text{ divise } a \\ 1 & \text{si } p \text{ ne divise pas } a \text{ et si } a \text{ est un résidu quadratique modulo } p \\ -1 & \text{si } p \text{ ne divise pas } a \text{ et si } a \text{ est un non résidu quadratique modulo } p \end{cases}$$

1.2 Le symbole de Jacobi

Le symbole de Jacobi est une généralisation du symbole de Legendre pour un entier n qui n'est pas nécessairement premier. Si a est un entier quelconque et n est un entier impair positif dont la décomposition en nombres premiers est $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$, le symbole de Jacobi $\left(\frac{a}{n}\right)$ est défini à l'aide des symboles de Legendre comme suit :

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \left(\frac{a}{p_2}\right)^{\alpha_2} \cdots \left(\frac{a}{p_k}\right)^{\alpha_k},$$

où chaque $\left(\frac{a}{p_i}\right)$, $i = 1, \ldots, k$ est un symbole de Legendre.

Remarque Si n est premier, le symbole de Jacobi et le symbole de Legendre sont égaux.

1.2.1 Propriétés du symbole de Jacobi

On suppose que n et m sont des entiers impairs positifs et a, b sont des entiers quelconques. Le symbole de Jacobi satisfait, entre autres, les propriétés suivantes :

1. Si $a \equiv b \mod n$ alors $\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right)$.
2. $\left(\frac{a}{n}\right) = \begin{cases} 0 & \text{si } \text{pgcd}(a,n) \neq 1 \\ \pm 1 & \text{si } \text{pgcd}(a,n) = 1 \end{cases}$
3. $\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$
4. $\left(\frac{1}{n}\right) = 1$
5. $\left(\frac{2}{n}\right) = (-1)^{\frac{n^2-1}{8}} = \begin{cases} 1 & \text{si } n \equiv 1 \mod 8 \text{ ou } n \equiv 7 \mod 8 \\ -1 & \text{si } n \equiv 3 \mod 8 \text{ ou } n \equiv 5 \mod 8 \end{cases}$
6. **Loi de réciprocité quadratique** Si $\text{pgcd}(m,n) = 1$ alors
$\left(\frac{m}{n}\right) = \left(\frac{n}{m}\right)(-1)^{\frac{m-1}{2}\frac{n-1}{2}} = \begin{cases} \left(\frac{n}{m}\right) & \text{si } n \equiv 1 \mod 4 \text{ ou } m \equiv 1 \mod 4 \\ -\left(\frac{n}{m}\right) & \text{si } n \equiv m \equiv 3 \mod 4. \end{cases}$

1.2.2 Calcul du symbole de Jacobi

Dans la pratique, il est très simple de calculer le symbole de Jacobi en utilisant les six propriétés ci-dessus. L'algorithme est, d'une certaine façon, analogue à l'algorithme d'Euclide pour calculer le pgcd de deux entiers.

Les étapes à suivre sont :

1. Réduire le numérateur modulo le dénominateur, en utilisant la propriété (1).
2. Extraire, s'il y en a, les facteurs 2 du nominateur en utilisant la propriété (3) et les évaluer selon la propriété (5).
3. Si le nominateur est 1, utiliser la propriété (4). Si le nominateur et le dénominateur ne sont pas premiers entre eux, alors la propriété (2) nous indique que le résultat est 0.
4. Autrement, le numérateur et le dénominateur, sont des entiers impairs positifs premiers entre eux. On utilise alors la propriété (6) pour échanger leur rôle et revenir à l'étape 1.

Un exemple. Calculer le symbol de Jacobi $\left(\frac{8721}{4235}\right)$.

$$\begin{aligned}
\left(\frac{8721}{4235}\right) &= \left(\frac{251}{4235}\right) \textit{ par la propriété (1), puisque } 8721 \equiv 251 \mod 4235 \\
&= -\left(\frac{4235}{251}\right) \textit{ par la propriété (6), puisque } 4235 \equiv 3 \mod 4 \textit{ et } 251 \equiv 3 \mod 4 \\
&= -\left(\frac{219}{251}\right) \textit{ par la propriété (1), puisque } 4235 \equiv 219 \mod 251 \\
&= \left(\frac{251}{219}\right) \textit{ par la propriété (6), puisque } 251 \equiv 3 \mod 4 \textit{ et } 219 \equiv 3 \mod 4 \\
&= \left(\frac{32}{219}\right) \textit{ par la propriété (1), puisque } 251 \equiv 32 \mod 219 \\
&= \left(\frac{2}{219}\right)^5 \textit{ par la propriété (3)} \\
&= (-1)^5 \textit{ par la propriété (5), puisque } 219 \equiv 3 \mod 8 \\
&= -1
\end{aligned}$$

Un autre exemple. Calculer le symbol de Jacobi $\left(\frac{541}{2011}\right)$.

$$\begin{aligned}
\left(\frac{541}{2011}\right) &= \left(\frac{2011}{541}\right) \textit{ par la propriété (6), puisque } 541 \equiv 1 \mod 4 \\
&= \left(\frac{388}{541}\right) \textit{ par la propriété (1), puisque } 2011 \equiv 388 \mod 541 \\
&= \left(\frac{2}{541}\right)^2 \left(\frac{97}{541}\right) \textit{ par la propriété (3)} \\
&= \left(\frac{97}{541}\right) \textit{ par la propriété (5)} \\
&= \left(\frac{541}{97}\right) \textit{ par la propriété (6), puisque } 541 \equiv 1 \mod 4 \\
&= \left(\frac{56}{97}\right) \textit{ par la propriété (1) puisque } 541 \equiv 56 \mod 97 \\
&= \left(\frac{2}{97}\right)^3 \left(\frac{7}{97}\right) \textit{ par la propriété (3)} \\
&= \left(\frac{7}{97}\right) \textit{ par la propriété (5), puisque } 97 \equiv 1 \mod 8 \\
&= \left(\frac{97}{7}\right) \textit{ par la propriété (6), puisque } 97 \equiv 1 \mod 4 \\
&= \left(\frac{6}{7}\right) \textit{ par la propriété (1), puisque } 97 \equiv 6 \mod 7 \\
&= \left(\frac{2}{7}\right)\left(\frac{3}{7}\right) \textit{ par la propriété (3)} \\
&= \left(\frac{3}{7}\right) \textit{ par la propriété (5), puisque } 7 \equiv 7 \mod 8 \\
&= -\left(\frac{7}{3}\right) \textit{ par la propriété (6), puisque } 7 \equiv 3 \mod 4 \textit{ et } 3 \equiv 3 \mod 4 \\
&= -\left(\frac{1}{3}\right) \textit{ par la propriété (1), puisque } 7 \equiv 1 \mod 3 \\
&= -1 \textit{ par la propriété (4)}
\end{aligned}$$

2 Le test de primalité de Solovay-Strassen

Euler a demontré que pour tout nombre premier impair p et pour tout entier a

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \mod p,$$

où $\left(\frac{a}{p}\right)$ est ici le symbole de Legendre.

Le test de Solovay-Strassen est basé sur cette propriété. Plus précisément, afin de determiner si un nombre impair n est premier ou composé, on peut choisir un entier a et tester si

$$\left(\frac{a}{n}\right) \equiv a^{\frac{n-1}{2}} \mod n, \tag{1}$$

où $\left(\frac{a}{n}\right)$ est maintenant le symbole de Jacobi. Si l'entier n est premier, alors le symbole de Jacobi est égal au symbole de Legendre et la congruence (1) est vérifiée pour tout entier a.

Si pour un entier a, l'égalité (1) n'est pas vérifiée, on peut alors conclure de façon certaine que le nombre n est composé. Sinon, il est probablement premier. Il peut être démontré que pour un entier impair composé, pour au moins la moitié des entiers $a \in \mathbb{Z}_n^*$, la congruence (1) est vérifiée. Ceci montre que la probabilité que le test "se trompe", c'est-à-dire qu'il declare qu'un entier n est premier alors qu'il

est composé, est autour de 1/2. Pour augmenter les chances que le test retourne un résultat correct, il suffit de répéter le test plusieurs fois, en choisissant à chaque fois une valeur de a différente. Plus le nombre de répétitions, k, augmente, plus la probabilité que le test se trompe (égale à $1/2^k$) diminue.

Le test que vous devez implémenter est décrit par l'algorithme 1. Pour cela, il vous faudra implémenter le calcul du symbole de Jacobi ainsi que l'algorithme d'exponentiation rapide (ou `square-and-multiply`) afin de calculer $a^{\frac{n-1}{2}}$ de façon efficace. Ce dernier algorithme est décrit dans la section suivante.

Algorithme 1 : `Test de Solovay-Strassen`

Données : Un entier n, dont on souhaite tester la primalité et le nombre k de répétitions
Résultat : `composé` si n est composé, sinon `probablement premier`

pour $i = 1, \ldots, k$ **faire**
  Choisir aléatoirement $a \in \{2, \ldots, n-1\}$
  $r \leftarrow \left(\frac{a}{n}\right)$
  si $r = 0$ *ou* $a^{\frac{n-1}{2}} \not\equiv r \mod n$ **alors**
    retourner `composé`
retourner `premier`

3 L'algorithme `square and multiply`

L'opération la plus coûteuse dans le test de Solovay-Strassen est l'exponentiation modulaire. Cette opération, si implémentée de manière naïve, peut faire exploser le temps du calcul. Par exemple, si on souhaite calculer $x^{2^{1024}}$ et procède en calculant d'abord $x^2 = x \cdot x$, ensuite $x^3 = x \cdot x^2$, jusqu'à $x^{2^{1024}} = x \cdot x^{2^{1024}-1}$, on va devoir effectuer $(2^{1024} - 1)$ multiplications !

Un algorithme qui permet de calculer l'exponentiation modulaire efficacement est la méthode `square and multiply`, qui utilise la décomposition binaire de l'exposant. Cette méthode, pour le cas de l'exponentiation modulaire est décrite par l'algorithme suivant.

Algorithme 2 : `square and multiply`

Données : Un entier a, le module n et un exposant $H = \sum_{i=0}^{t} h_i 2^i$, avec $h_i \in \{0, 1\}$ et $h_t = 1$
Résultat : $a^H \mod n$

$r \leftarrow a$
pour $i = t-1, \ldots, 0$ **faire**
  $r \leftarrow r^2 \mod n$;
  si $h_i = 1$ **alors**
    $r \leftarrow r \cdot a \mod n$;
Sorties : r

Pour calculer par exemple $x^{26} = x^{11010_2}$ (on ne s'intéresse pas ici aux opérations modulaires), les étapes à effectuer sont les suivantes :

`SQ`	$x \cdot x = x^2$
`MUL`	$x \cdot x^2 = x^3$
`SQ`	$x^3 \cdot x^3 = x^6$
`SQ`	$x^6 \cdot x^6 = x^{12}$
`MUL`	$x \cdot x^{12} = x^{13}$
`SQ`	$x^{13} \cdot x^{13} = x^{26}$

4 La bibliothèque `GNU MP`

Le langage `C` possède plusieurs types pour représenter les nombres entiers. Cependant, tous ces types ont une précision fixe et ne peuvent pas dépasser un certain nombre d'octets. Le type le plus grand est

`long long int` qui peut contenir des entiers d'une taille maximale de 64 bits. Or, tous ces types sont beaucoup trop courts pour les applications cryptographiques qui nécessitent la manipulation des données d'au moins 512 bits.

`GNU MP`, pour `GNU Multi Precision`, souvent appelée `GMP` est une bibliothèque `C/C++`, de calcul multiprécision sur des nombres entiers, rationnels et à virgule flottante qui permet en particulier de manipuler des très grands nombres.

Vous pouvez télécharger une archive contenant cette bibliothèque à l'adresse

https://gmplib.org/#DOWNLOAD

et le manuel d'utilisation à l'adresse

https://gmplib.org/gmp-man-6.1.2.pdf

La page 3 de ce guide vous explique comment installer cette bibliothèque sous `Linux`. La bibliothèque a été principalement conçue pour des systèmes `Unix`, l'installation et la configuration pour `Windows` est plus complexe. `Google` pourrait vous aider avec cela, si vous insistez pour faire le DM sous `Windows`. Notez que je testerai vos programmes sous `Ubuntu`.

Vos programmes doivent inclure le fichier `'gmp.h'` et la compilation doit se faire avec l'option `'-lgmp'`, par exemple

```
gcc monprogramme.c -lgmp
```

Manipulation de nombres entiers. Le type d'un entier avec `GMP` est `mpz_t`. Pour déclarer un entier `n` vous devez écrire

```
mpz_t n;
```

Avant d'affecter un nombre entier à la variable `n`, cette variable doit être initialisée, en utilisant une des fonctions d'initialisation, par exemple :

```
mpz_init(n);
```

Si vous n'avez plus besoin de cette variable, vous devez alors la libérer :

```
mpz_clear(n);
```

La bibliothèque contient un très grand nombre de fonctions, (`mpz_add(), mpz_sub(), mpz_div()`,...) qui permettent d'effectuer des opérations classiques et moins classiques sur les nombres entiers. Le manuel de `GMP` contient la description et le mode d'utilisation de toutes les fonctions dont vous aurez besoin. Vous pouvez visiter la page de `Wikipedia`,

http://en.wikipedia.org/wiki/GNU_Multiple_Precision_Arithmetic_Library

pour voir un exemple d'un programme simple.

5 À vous maintenant

Vous devez implémenter le test de Solovay-Strassen décrit dans la section 2 en utilisant la librairie `GMP`. L'utilisateur doit fournir au programme l'entier n à tester ainsi que l'entier k, indiquant le nombre d'itérations souhaité. L'algorithme `square and multiply` doit être implémenté pour l'exponentiation modulaire et il faudra aussi programmer le calcul du symbole de Jacobi.

Pour que la demarche ne soit pas trop simple, vous n'êtes pas autorisés à utiliser les fonctions de la libairie `GMP` suivantes : `mpz_probab_prime_p`, `mpz_nextprime`, `mpz_gcd`, `mpz_gcd_ui`, `mpz_gcdext`, `mpz_powm`, `mpz_powm_ui`, `mpz_powm_sec`, `mpz_pow_ui` et `mpz_ui_pow_ui, mpz_jacobi, mpz_legendre`, et en général aucune fonction de cette librairie qui rendrait la tâche trop facile.

Pour tester vos programmes voici une page qui contient un grand nombre de nombres premiers

http://en.wikipedia.org/wiki/List_of_prime_numbers

Testez également vos programmes avec certains des nombres pseudo-premiers fournis ici

http://fr.wikipedia.org/wiki/Nombre_pseudo-premier

Votre programme doit être capable de tester des nombres premiers vraiment grands. Pour cela, un fichier avec quelques très grands nombres premiers vous est également fourni. Vérifiez que votre programme fonctionne bien pour tous ces nombres.

Vous devez m'envoyer vos programmes par mail. Pour faciliter la démarche, j'attends de vous une archive compressée, nommée de vos prénoms et noms, contenant le code source. Celui-ci doit être bien commenté et propre. Si votre code contient plus d'un fichier, un `makefile` est alors nécessaire. Dans tous les cas, vous devez fournir un fichier `README.txt` expliquant comment compiler et tester votre code. Vous pouvez travailler **individuellement** ou **par équipe de deux personnes au plus**. Bien évidement, chaque équipe devra travailler seule.

La date limite pour m'envoyer vos programmes est le **lundi 22 avril à 23h**. Un point de pénalité sera attribué à chaque heure de retard.