Tests de primalité

Référence : *Cours d'algèbre*, M. Demazure.

La sécurité des cryptosystèmes RSA et El Gamal repose sur notre capacité à fournir de grands nombres premiers au hasard, en un temps négligeable devant celui qu'il faudrait pour les factoriser (ou plus exactement, pour factoriser des nombres composés du même ordre de grandeur.) Il est aujourd'hui préconisé, pour RSA, de recourir à des nombres premiers ayant entre 1024 et 2048 bits.

Questions :

1. Quelle est la proportion de nombres premiers dans cet intervalle ?
2. En supposant que l'on est capable de tirer aléatoirement, et de manière indépendante, des entiers dans cet intervalle, au bout de combien de tentatives peut-on espérer obtenir un nombre premier ?

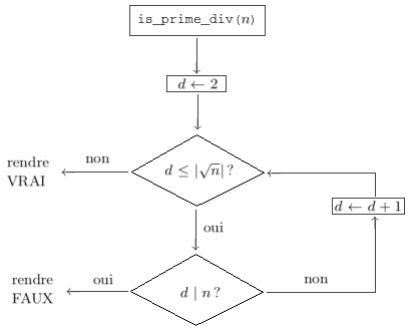
Ce TP se divise en deux parties : dans un premier temps, on étudie les performances des algorithmes naïfs et le principe général d'un test probabiliste ; puis l'on présente deux tests avancés : Fermat et Solovay-Strassen, qui permettent de détecter, avec une certitude arbitrairement grande, qu'un nombre est premier. Enfin, une fois que l'on dispose d'un candidat pressenti à la primalité, on peut éventuellement la garantir au moyen d'un *certificat* plus coûteux mais déterministe.

Le but du TP est de mettre en place un générateur de grands nombres premiers.

1 Etude des algorithmes naïfs

Détecter la primalité via la non-divisibilité

On considère l'algorithme suivant :



1. Justifier que cet algorithme termine. Combien de fois au plus passe-t-on dans la boucle ? Quelle est la complexité au pire ?
2. Justifier que l'algorithme détecte bien tous les nombres premiers, et ceux-là seulement.
3. L'implémenter et le tester sur 7, 561, 1991, 1993, $10^8 + 7$ et $10^{10} + 19$. Afficher les temps de calcul pour les deux derniers tests.
4. Ecrire une fonction `nb_etapes_moyen_div(N,k)` qui compte le nombre de passages dans la boucle de k entiers choisis aléatoirement entre 2 et N et en déduit le nombre moyen de passages. Tester et illustrer (par exemple via `bar_chart`) pour $N = 100000$ et $k = 1000$.

Détecter la primalité via la coprimalité

On propose d'améliorer l'algorithme précédent en ne testant plus la divisibilité de n par d, mais leur coprimalité. Par exemple, bien que ne divisant pas 91, 26 permet cependant de garantir que 91 est composé puisque $\text{pgcd}(26, 91) = 13$.

On appellera désormais *témoin de non-coprimalité de* n tout entier $1 < a < n$ tel que $\text{pgcd}(a, n) \neq 1$. Il y a en général bien plus de témoins de non-coprimalité que de diviseurs.

1. En reprenant les questions précédentes, écrire les fonctions `is_prime_coprime(n)` et `nb_etapes_moyen_div(N,k)`. De nouveau, tester et illustrer pour $N = 100000$ et $k = 1000$. Que remarquez-vous ? Comment l'expliquez-vous ?
2. Pourquoi le système RSA ne peut-il pas s'appuyer, pour fabriquer ses clés, sur l'un ou l'autre de vos algorithmes `is_prime_div(n)` / `is_prime_coprime(n)` ?

Un premier pas vers les algorithmes probabilistes

Un algorithme est dit *probabiliste* si son comportement dépend à la fois des données du problème et de valeurs produites par un générateur de nombres aléatoires. En particulier un même algorithme probabiliste, exécuté deux fois sur le même jeu de données, peut rendre deux réponses différentes du fait de valeurs aléatoires différentes.

Les tests de primalité rapides que nous allons étudier sont des algorithmes probabilistes. Ils s'appuient sur une condition $\mathcal{P}$ nécessaire (mais pas toujours suffisante) de primalité - par exemple être co-premier avec tous les entiers $1 < a < n$.

Il s'agit alors de tirer au hasard un entier et de tester la propriété sur cet entier. S'il la vérifie, on dira que n est "probablement premier" ; s'il la réfute, on parlera de *témoin de non-primalité pour la propriété* $\mathcal{P}$. L'existence d'un tel témoin garantit que n est composé.

Nous allons construire nos premiers tests de primalité à partir des propriétés :
- $\mathcal{P}_{div}$: p premier $\Rightarrow \forall\ 1 < d < n,\ d$ ne divise pas n.
- $\mathcal{P}_{coprime}$: p premier $\Rightarrow \forall\ 1 < a < n,\ \text{pgcd}(a, n) = 1$.

1. Ecrire les fonctions booléennes `temoin_div(n,d)` et `temoin_coprime(n,a)`.
2. Ecrire deux algorithmes `is_prime_rand_div(n)` et `is_prime_rand_coprime(n)` qui, à partir de 10 entiers entre 2 et $n-1$ tirés au hasard, décident si n est "probablement premier" ou "composé" à partir des propriétés respectives $\mathcal{P}_{div}$ et $\mathcal{P}_{coprime}$. Les tester sur 64 et 93.
3. Ecrire une fonction qui tire au hasard 1000 entiers entre 2 et 100000 et compte le nombre de "probablement premiers" obtenus via chacun des deux tests ci-dessus avec le nombre d'entiers effectivement premiers (utiliser la fonction SAGE `is_prime`). Illustrer d'un diagramme. Calculer le taux d'erreur de chacune des méthodes.

Tout l'enjeu des tests de primalité est de contrôler (puis minimiser) l'apparition de faux-positifs.

2 Tests de primalité avancés

Test de Fermat

On considère la propriété suivante :
$\mathcal{P}_{Fermat}$: p premier $\Rightarrow \forall\ a \in \mathbb{Z}$ premier avec $n,\ a^{p-1} = 1\,[p]$.

1. Donner une preuve élémentaire de cette propriété.
2. Ecrire une fonction `temoin_fermat(n,a)`.
   En déduire un algorithme `is_prime_fermat(n)` qui tire au hasard 3 entiers entre 1 et $n-1$ et teste s'ils sont témoins de non-coprimalité ou témoins de Fermat. Le tester sur 7,

561, 1991, 1993, 10^8+7 et $10^{10}+19$. Afficher les temps de calcul pour les deux derniers tests.

3. Ecrire une fonction qui tire au hasard 1000 entiers entre 2 et 100000 et compte le nombre de "probablement premiers" obtenus via le test de Fermat avec le nombre d'entiers effectivement premiers. Calculer le taux d'erreur.
4. [Important] Montrer que si $n \geq 2$ admet au moins un témoin de Fermat, alors n en admet au moins $\varphi(n)/2$, où φ désigne l'indicatrice d'Euler. En déduire que, dans ce cas, que n admet au moins $n/2$ témoins inférieurs à n.
5. Que pouvez-vous dire sur la probabilité que votre algorithme `is_prime_fermat` produise un faux-positif ? Calculer empiriquement cette probabilité pour 561 puis 6601.

Hélas, il existe des entiers composés qui vérifient le petit théorème de Fermat, et qui n'admettent donc pas de témoin de Fermat. Pour ces nombres là, l'algorithme est aussi mauvais (et imprévisible) que `is_prime_coprime`. On les appelle *nombres de Carmichael*. Nous ne savons que depuis 1994 qu'il en existe une infinité ! Le plus petit nombre de Carmichael est 561.

Test de Solovay-Strassen

On rappelle qu'étant donné p un nombre premier, le symbole de Legendre $(\frac{a}{p})$ vaut 0 si a est multiple de p, 1 si a est un carré non nul modulo p et -1 sinon. Si à présent n désigne un entier impair supérieur à 3, que l'on peut toujours écrire $n = p_1^{\alpha_1}...p_k^{\alpha_k}$, le symbole de Jacobi $(\frac{a}{n})$ est égal au produit des symboles de Legendre :

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \ ... \ \left(\frac{a}{p_k}\right)^{\alpha_k}.$$

La loi de réciprocité quadratique permet de calculer effectivement et efficacement les symboles de Jacobi (et donc en particulier de Legendre.)

Nous considérons à présent le critère d'Euler, sur lequel s'appuie le test de Solovay-Strassen : $\mathcal{P}_{SS}$: $p > 2$ premier $\Rightarrow \ \forall\, 0 \leq a \leq p-1,\ (\frac{a}{p}) = a^{(p-1)/2}\ [p]$.

1. Démontrer le critère d'Euler.
2. Pour rappel, énoncer la loi de réciprocité quadratique.

On dira que a est *témoin d'Euler* si $\text{pgcd}(a,n) = 1$ et $(\frac{a}{n}) \neq a^{(n-1)/2}\ [n]$.

Solovay et Strassen ont montré, via une caractérisation des nombres de Carmichael, que tous les nombres composés admettent au moins un témoin d'Euler. Comme précédemment, on en déduit qu'ils en admettent alors au moins $\varphi(n)/2$, puis que **tout nombre composé admet au moins $n/2$ témoins de non-primalité.**

5. A l'aide de la fonction SAGE `jacobi_symbol(a,n)`, écrire `temoin_euler(n,a)`.
6. Ecrire un algorithme `is_prime_SS(n,nb_essais)` qui tire *nb_essais* entiers entre 3 et $n-1$ puis teste s'il s'agit de témoins de non-coprimalité ou d'Euler. Tester votre fonction pour *nb_essais*= 3 sur 7, 561, 1991, 1993, 10^8+7 et $10^{10}+19$. Afficher les temps de calcul pour les deux derniers tests..
7. Ecrire une fonction qui tire au hasard 1000 entiers entre 2 et 100000 et compte le nombre de "probablement premiers" obtenus via le test de Solovay-Strassen avec le nombre d'entiers effectivement premiers. Calculer le taux d'erreur.
8. Que pouvez-vous dire sur la probabilité que votre algorithme `is_prime_SS` produise un faux-positif ? Est-elle vraie pour 6601 ?

Question finale : Ecrire un algorithme `random_prime(min,max,eps)` qui tire un nombre entre *min* et *max* qui est premier avec probabilité supérieure à $1-eps$. Exhiber de grands nombres premiers.

♠ Pour aller plus loin...

— Caractéristation des nombres de Carmichael et preuve du théorème de Solovay-Strassen
— Test de primalité de Rabin-Miller
— Certificats de primalité...
— Problèmes NP et co-NP... (lire par exemple le chapitre sur le sujet dans *Introduction à l'algorithmique*, Cormen-Leiserston-Stein-Rivest.)