TD génération de (grands) nombres premiers

I Principe général

Dans ce TD, seront abordés des algorithmes du type Monte Carlo, c'est à dire qu'ils ne donnent pas exactement la réponse attendue, mais une bonne approximation (dans le cas des nombres premiers, les algorithmes ne diront pas qu'un nombre est premier, mais qu'il est probablement premier). Ils ont l'avantage d'être rapides et peuvent généralement diminuer l'incertitude à volonté.

test de primalité Pour déterminer si oui ou non un nombre n est premier, il suffit de vérifier que $\mathbb{Z}\backslash n\mathbb{Z}$ est un corps, ou de vérifier qu'il n'existe aucun entier a non multiple de n tel que $a^{n-1} \not\equiv 1 \ \ [n]$. C'est pourquoi, par la suite la majorité des opérations seront effectuées modulo n.

II fonctions utiles

1. Créer une fonction `puissance_modulaire` prenant en argument des entiers a, e, n et renvoyant a^e mod n (sans utiliser l'opérateur "**").
2. En constatant que l'exposant peut s'écrire sous forme de produit de puissances de deux (son écriture binaire), et en sachant que $a^{2^k} = (a^{2^{k-1}})^2$, écrire une fonction `puissance_modulaire_rapide` effectuant la même tâche que la fonction précédente.
3. Étudier rapidement la complexité de ces deux fonctions, laquelle est la plus rapide ?
4. À l'aide de la fonction `randint` du module `random` qui a pour argument deux entiers a et b qui renvoie un entier aléatoire dans $[\![a \ ; \ b-1]\!]$, écrire une fonction `entier_aleatoire` prenant en argument le nombre de bits du nombre à générer et renvoyant un entier aléatoire de la taille indiquée.
5. À l'aide de la question précédente, écrire une fonction renvoyant un nombre impair (il aura plus de chance d'être premier qu'un nombre pair) aléatoire de taille donnée.

III Utilisation du théorème de Fermat (le petit)

Ce théorème peut s'énoncer comme suit :
Pour p premier, quel que soit a non multiple de p (tel que $a \not\equiv 0 \ \ [p]$) alors

$$a^{p-1} \equiv 1 \ \ [p]$$

6. Créer une fonction `est_compose` prenant en argument deux entiers a et n et renvoyant, à l'aide du théorème de Fermat, un `boolean` traduisant le fait que n est composé.
7. Créer alors une fonction `est_potentiellement_premier` qui prend un entier n et qui, à l'aide de la question précédente, renvoie un `boolean` traduisant la primalité de n. (on pourra se limiter à tester n avec les entiers 2, 3, 5 et 7)

Toutefois, cette méthode a des limites. On pourrait tester plus de nombre pour vérifier la primalité de n (idéalement tous les nombres premiers $< n$) mais cela montre ses limites. En effet, certains nombres composés remplissent quand même cette condition de primalité, ces nombres sont appelés *nombres de Carmichaël*, 561 est un de ces nombres.

IV p premier $\Leftrightarrow \mathbb{Z}\backslash p\mathbb{Z}$ est un corps

En constatant que, dans un corps, l'équation $x^2 = 1$ n'a que deux solutions : ± 1, il est alors possible de déterminer si $\mathbb{Z}\backslash n\mathbb{Z}$ est un corps en regardant les solutions de cette équation.

par exemple : $3^2 \equiv 1 \ [8]$ or $3 \neq \pm 1$, on sait alors que $\mathbb{Z}\backslash 8\mathbb{Z}$ n'est pas un corps et donc que `8` n'est pas premier (quelle surprise !).

8. Écrivez alors une fonction `est_compose_2` qui prend en argument n et a ($a \in [\![2\ ;\ p-2]\!]$) et, à l'aide du raisonnement précédent, renvoie un `boolean` traduisant la composition de n.
9. En poussant le raisonnement, on peut calculer le carré du carré que l'on vient de calculer, puis le carré du carré du carré etc... Jusqu'à avoir dépassé a^{n-1}. Écrivez alors une fonction `est_compose_3` traduisant ce fait.

optimisation, algorithme de Miller Rabin En constatant que, quel que soit $n > 1$, on dispose de deux uniques entiers s et d (d impair) tels que $n - 1 = 2^s d$ (pour $s = V_2(n-1)$ (le nombre de 2 dans la décomposition en produit de facteurs premiers) et $d = \frac{n-1}{2^s}$). On peut alors réécrire le théorème de Fermat :

Pour p premier, $\forall a \in [\![1\ ;\ p-1]\!]$, s et d tel que $p - 1 = 2^s d$

$$a^{2^s d} \equiv 1 \ [p]$$

$$\Leftrightarrow a^{d^{2^s}} \equiv 1 \ [p]$$

$$\Leftrightarrow \textit{pour} \ \ x = a^d : x^{2^s} \equiv 1 \ [p]$$

10. Écrivez une fonction prenant n comme argument et renvoyant s et d tels que $n - 1 = 2^s d$.
11. L'algorithme de Miller Rabin se présente comme suit :
    Pour n impair, on considère s et d tels que $n - 1 = 2^s d$, on prend alors un entier aléatoire $a \in [\![2\ ;\ n-2]\!]$, on calcule a^d modulo n. puis les s carrés successifs (jusqu'à avoir $a^{2^s d} = a^{n-1}$).

    — Si $a^d = \pm 1 \ [n]$ il n'y a pas de contradiction et n est peut être premier.
    — Sinon, on calcule les s carrés successifs.
        — Si l'un des carrés vaut 1 alors que le carré précédent n'est pas -1 (sans compter le cas ou $a^d = 1 \ [n]$), alors $\mathbb{Z}\backslash n\mathbb{Z}$ n'est pas un corps et donc n est composé.
        — Si le dernier carré n'est pas 1, n est composé (le théorème de Fermat n'est pas vérifié)
        — Si le dernier carré vaut 1 alors n est potentiellement premier.

Écrivez la fonction associée prenant deux entiers n et a et renvoyant un `boolean` traduisant la composition de n.

12. Il se trouve que certains des entiers de $[\![2\ ;\ p-2]\!]$ ne permettent pas de trancher sur la composition de n, alors même que celui-ci est composé (ils sont appelés "menteur fort"). Toutefois, la probabilité qu'à l'issu d'un test, l'algorithme déclare un nombre premier alors qu'il ne l'est pas est inférieur à $1/4$ ainsi, en effectuant k fois le test avec des nombres différents, la probabilité d'erreur devient négligeable (inférieure 4^{-k}). Écrivez donc une fonction `est_premier` prenant en argument un entier n et renvoyant un `boolean` traduisant la potentielle primalité de n (on poura effectuer 25 fois le test précédent pour avoir un taux d'erreur de moins de 10^{-14} %).
13. Finalement, à l'aide de toute ces questions, écrivez une fonction `random_prime` prenant en argument la taille du nombre premier désiré et renvoyant un nombre premier aléatoire de cette taille.