AIT MIMOUN Yasmine

TRAORE Madina

Rapport de projet

Tests de primalité

4I900

Master d'Informatique

Sorbonne Université

Année universitaire 2018-2019

Sommaire

Au cours de ce projet nous allons implanter des algorithmes permettant de tester la primalité d'un entier : d'abord un test naïf qui ne se trompe jamais puis deux tests probabilistes (Test de Fermat et Test de Rabin et Miller) qui se trompent avec une certaine probabilité en affirmant qu'un entier est premier alors qu'il ne l'est pas. Nous tenterons d'estimer le taux d'erreur de ces tests probabilistes en comparant leurs réponses aux réponses fournies par le test naïf ne se trompant jamais. L'ensemble des fonctions utilisées seront codées en Python.

1 Exercice 1 : Arithmétique dans $\mathbb{Z}_N$

Écrivons tout d'abord des fonctions permettant de travailler modulo un entier N de taille arbitraire.

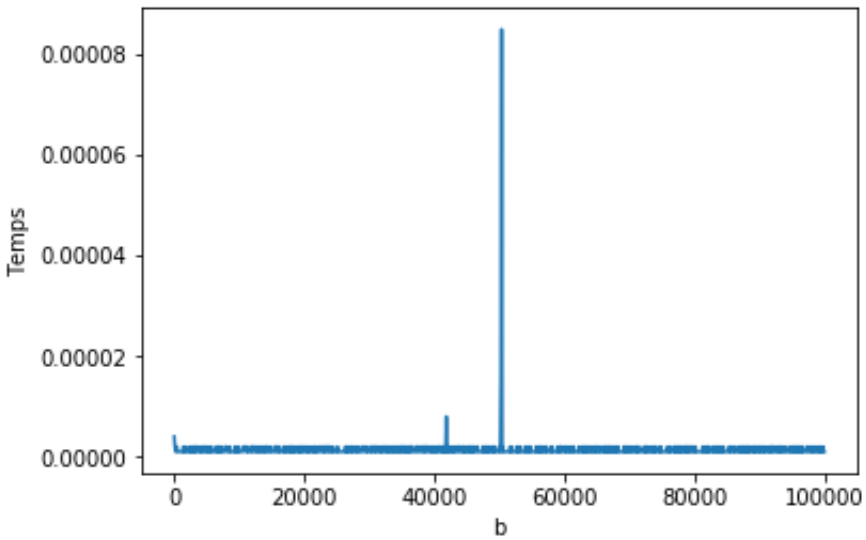
1.1 PGCD

La première fonction que nous souhaitons implanter est la fonction **my_pgcd** permettant de calculer le pgcd de deux entiers a et b donnés. Pour ce faire, on utilise l'algorithme d'Euclide décrit ci-dessous et dont l'idée est d'effectuer des divisions successives puis de s'arrêter dès lors que le reste de la division est nul : à cet instant on a alors trouvé le plus grand diviseur commun à a et b.

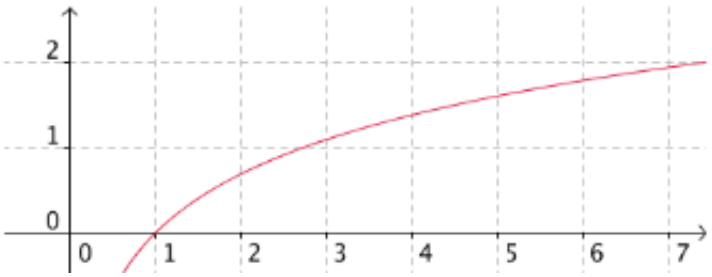
Algorithme PGCD

Entrée : $a, b \in \mathbb{N}$
Sortie : $pgcd(a, b)$
  tant que $b \neq 0$
    $a \leftarrow b$
    $b \leftarrow a \pmod{b}$
  fin tant que
  retourner a

Déterminons la compléxité de notre fonction **my_pgcd**. Pour cela on trace la courbe du temps d'éxecution de la fonction en fonction de b et on fait varier a entre b et $b+x$ avec x un entier choisi aléatoirement entre 1 et 15.
Ici nous ne sommes pas parvenues à obtenir une courbe cohérente et cela même en tentant de supprimer les valeurs aberrantes. On obtient en effet une courbe ayant la forme suivante :



Or l'algorithme d'Euclide ayant une complexité en $\mathcal{O}(log(b))$ (puisque l'on fait des divisions successives jusqu'à obtenir $b = 0$), on devrait observer une courbe ayant la forme suivante :



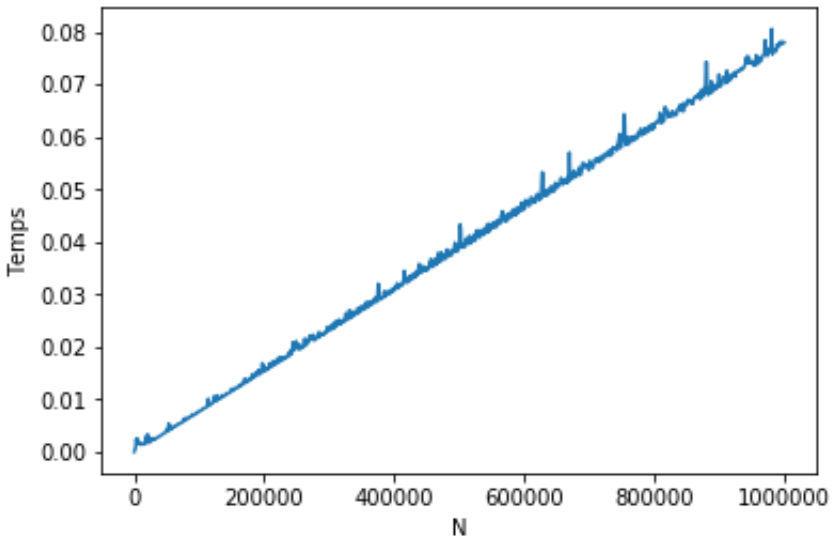
1.2 Inverse d'un entier modulo un autre entier

La deuxième fonction que nous voulons écrire est la fonction **my_inverse** renvoyant, s'il existe, l'inverse d'un entier a donné modulo un autre entier N. Pour le trouver, il suffit de parcourir tous les entiers b allant de 1 à N et de vérifier à chaque fois si $a \times b \equiv 1 \pmod{N}$. Si tel est le cas alors b est l'inverse modulo N de a. S'il n'existe pas de tel entier, alors a n'est pas inversible modulo N.

Algorithme Inverse modulo un entier

Entrée : $a, N \in \mathbb{N}$
Sortie : b tel que b est l'inverse de a modulo N, rien sinon
    pour b allant de 1 à N **faire**
        si $a \times b \equiv 1 \pmod{N}$
            retourner b
        fin si
    fin pour

Déterminons à présent la compléxité de notre fonction **my_inverse**. Pour cela on trace la courbe du temps d'éxecution de la fonction en fonction de N :



On obtient une droite ce qui signifie que l'algorithme est en $\mathcal{O}(N)$. Cela est effectivement le cas puisque l'on parcourt les N-1 entiers de 1 à N-1.

1.3 Exponentiation rapide

On souhaite maintenant écrire une fonction **my_expo_mod** qui, étant donnés trois entiers g,n et N calcule $g^n \pmod{N}$ avec une méthode d'exponentiation discrète binaire. Pour cela, on écrit tout d'abord

une fonction **binaire** permettant d'obtenir les coefficients $a_i \in \{0,1\}$ tels que $n = \sum_{i=0}^{l-1} a_i 2^i$. On calcule ensuite $g^n \pmod N$ grace à l'algorithme présenté dans l'énoncé.

Un exemple d'éxuction de la fonction :

```
In [900]: g = 2
          n = 3
          N = 3
          print(my_expo_mod(g,n,N))

          2
```

2 Exercice 2 : Test naïf et recherche des nombres de Carmichael

Dans cette partie, on cherche à montrer que les nombres de Carmichael sont relativement rares.

2.1 Test naïf de primalité

Premièrement, écrivons une fonction **first_test** permettant de déterminer de façon certaine si un entier est premier ou non. Pour ce faire, on implante l'algorithme déterministe suivant :

Algorithme Test naïf de primalité

Entrée : $N \in \mathbb{N}$
Sortie : $VRAI$ si N est premier, $FAUX$ sinon
  pour i allant de 2 à $\lfloor\sqrt{N}\rfloor$ **faire**
    si $N \equiv 0 \pmod i$
      retourner $FAUX$
    fin si
  fin pour
  retourner $VRAI$

Complexité : On parcourt au plus tous les entiers de 2 à $\lfloor\sqrt{N}\rfloor$, soit $\lfloor\sqrt{N}\rfloor - 2$ entiers. L'algorithme est donc en $\mathcal{O}(\sqrt{N})$ (autrement dit en $\mathcal{O}(N^{\frac{1}{2}})$).

On souhaite dénombrer les nombres premiers inférieurs à 10^5. On écrit pour cela une fonction **nb_premiers_inf_a** qui utilise **first_test** et on obtient le résultat suivant :

```
In [17]: nb_premiers_inf_a(10**5)
Out[17]: 9592
```

Il existe donc 9592 nombres premiers inférieurs à 10^5.

2.2 Génération de nombres de Carmichael

Nous souhaitons lister les nombres de Carmichael inférieurs à 10^5.

On écrit alors une fonction **carmichael** déterminant si un entier n donné est un nombre de Carmichael ou non puis une fonction **gen_carmichael_inf_a** qui utilise **carmichael** et renvoie les nombres de Carmichael inférieurs à un entier N donné.

On obtient les nombres de Carmichael suivants :

```
In [289]: gen_carmichael_inf_a(10**5)
561
1105
1729
2465
2821
6601
8911
10585
15841
29341
41041
46657
52633
62745
63973
75361
```

À présent, écrivons une fonction **gen_carmichael** permettant de génerer un nombre de Carmichael avec 3 facteurs premiers. On génere d'abord 3 nombres premiers p,q et r en choisissant à chaque fois aléatoirement un nombre compris entre deux entiers a et b donnés puis en testant s'ils sont premiers ou non grace à la fonction **first_test** écrite précedemment. On teste enfin si le nombre $n = pqr$ est un nombre de Carmichael grace à notre fonction **carmichael**.

Le plus grand nombre de Carmichael trouvé avec notre implémentation de **gen_carmichael** est 294409 (trouvé en 4 minutes et 17 secondes).

```
In [333]: gen_carmichael(35,200)
294409 = 37 x 73 x 109

Out[333]: 294409
```

2.3 Nombres de Carmichael particuliers

Montrons qu'il n'existe qu'un nombre fini de nombres de Carmichael de la forme pqr avec $p < q < r$ trois nombres premiers tels que p est fixé.

Soit n un nombre de Carmichael tel que $n = pqr$ avec $p < q < r$ trois nombres premiers tels que p est fixé.

Montrons tout d'abord qu'il existe un entier $h \in \{2, ..., p-1\}$ *tel que* $(pq-1) = h(r-1)$.

Preuve

On a :
$n = pqr$ d'où $n - 1 = pqr - 1 = pqr - pq + pq - 1 = pq(r-1) + (pq-1)$
Donc $pq - 1 = (n-1) - pq(r-1)$.
$r|n$ donc d'après le critère de Korselt on a $(r-1)|(n-1)$ autrement dit : $n - 1 \equiv 0 \pmod{(r-1)}$.
On a donc :
$pq - 1 \equiv (n-1) - pq(r-1) \equiv -pq(r-1) \equiv 0 \pmod{(r-1)}$
Il existe donc $h \in \mathbb{N}$ tel que $(pq-1) = h(r-1)$.

Supposons que $h = 0$.
On a alors :
$(pq-1) = h(r-1) \Leftrightarrow pq - 1 = 0$ d'où $pq = 1 \Rightarrow p = \frac{1}{q}$.
Or, p est un entier.

CONTRADICTION
Donc $h \neq 0$.

Supposons que $h = 1$.
On a alors :
$(pq-1) = h(r-1) \Leftrightarrow pq - 1 = r - 1$ d'où $pq = r \Rightarrow p = \frac{r}{q}$.
Or, r et q sont premiers donc $\frac{r}{q} \notin \mathbb{N}$.
CONTRADICTION car p est un entier.
Donc $h \neq 1$.
D'où : $h \geq 2$.

Supposons que $h = p$.
On a alors :
$(pq-1) = h(r-1) \Leftrightarrow pq - 1 = p(r-1) \Leftrightarrow pq - 1 = pr - p \Leftrightarrow p(q-r+1) = 1 \Leftrightarrow q - r + 1 = \frac{1}{p}$.
ABSURDE étant donné que $p, q, 1 \in \mathbb{N}$.

Supposons maintenant que $h \geq p + 1$.
On a alors :
$h(r-1) \geq (p+1)(r-1) \geq pr - p + r - 1$.
Donc $pq - 1 \geq pr - p + r - 1 \Rightarrow pq \geq pr - p + r \geq pq - p + r$. (car $r > q$)
Donc $pq \geq pq + r - p$.
ABSURDE étant donné que $r > p$ et r et p sont premiers donc $r - p \geq 2$.

Donc $h \in \{2, ..., p-1\}$.

Il existe donc bien $h \in \{2, ..., p-1\}$ tel que $(pq-1) = h(r-1)$.

Montrons à présent qu'il existe un entier k tel que $(hk - p^2)(q-1) = (p+h)(p-1)$.

Preuve

On a :
$pr - 1 = pr - p + p - 1 = p(r-1) + (p-1)$ d'où $h(pr-1) = ph(r-1) + h(p-1) = p(pq-1) + h(p-1)$ (d'après la question 1.).
Or, d'après le critère de Korselt on a $n - 1 \equiv 0 \pmod{(q-1)}$.
Or $n - 1 \equiv pqr - 1 \equiv pqr - pr + pr - 1 \equiv pr(q-1) + (pr-1) \pmod{(q-1)}$ donc $pr - 1 \equiv 0 \pmod{(q-1)}$.
Il existe donc $k \in \mathbb{N}$ tel que $pr - 1 = k(q-1)$.
On en déduit :
$h(pr-1) = p(pq-1) + h(p-1)$
$\Leftrightarrow hk(q-1) = p(pq-1) + h(p-1)$
$\Leftrightarrow hk(q-1) = p^2q - p + h(p-1)$
$\Leftrightarrow hk(q-1) = p^2q - p^2 + p^2 - p + h(p-1)$
$\Leftrightarrow hk(q-1) = p^2(q-1) + p(p-1) + h(p-1)$
$\Leftrightarrow hk(q-1) - p^2(q-1) = (p+h)(p-1)$
$\Leftrightarrow (hk - p^2)(q-1) = (p+h)(p-1)$

Il existe donc bien $k \in \mathbb{N}$ tel que $(hk - p^2)(q-1) = (p+h)(p-1)$.

Conclusion

p est fixé et $h \leq p - 1$ donc il existe $N_1 \in \mathbb{N}$ tel que $(p+h)(p-1) \leq N_1$.
Donc $(hk - p^2)(q-1) \leq N_1$ (d'après la question 2).
Il existe donc $N_2 \in \mathbb{N}$ tel que $q \leq N_2$ et donc $N_3 \in \mathbb{N}$ tel que $(pq-1) \leq N_3$.
D'après la question 1., on a alors aussi : $h(r-1) \leq N_3$.
Donc il existe $N_4 \in \mathbb{N}$ tel que $r \leq N_4$.
q et r sont donc tous deux bornés (i.e. ont un nombre fini de valeurs possibles) : il existe donc un nombre fini de nombres de Carmichael de la forme pqr avec $p < q < r$ trois nombres premiers tels que p est fixé.

On utilise la fonction **gen_carmichael_p_fixe** pour lister les nombres de Carmichael de la forme $3qr$ et de la forme $5qr$ avec q et r premiers :

```
In [331]: gen_carmichael_p_fixe(3,1,20)
          561 = 3 x 11 x 17

Out[331]: [561]

In [332]: gen_carmichael_p_fixe(5,1,100)
          1105 = 5 x 13 x 17

          2465 = 5 x 17 x 29

          10585 = 5 x 29 x 73

Out[332]: [1105, 2465, 10585]
```

3 Exercice 3 : Test de Fermat

Le test de Fermat est un test probabiliste permettant de déterminer si un nombre est composé ou possiblement premier. L'algorithme est le suivant :

Algorithme Test de Fermat

Entrée : $N \in \mathbb{N}$
Sortie : *COMPOSÉ* si N est composé, *POSSIBLEMENT PREMIER* sinon
$a \xleftarrow{R} \{2, N-1\}$
$b \leftarrow a^N \pmod N$
if $b - a \not\equiv 0 \pmod N$
  retourner *COMPOSÉ*
fin si
retourner *POSSIBLEMENT PREMIER*

Tentons d'estimer la probabilité d'erreur de notre fonction **test_fermat** implémentant cet algorithme en regardant si elle répond correctement lorsqu'on lui donne aléatoirement des nombres de Carmichael (à l'aide de la fonction **gen_carmichael**), des nombres composés (grâce à **gen_compose**) ou des nombres tirés aléatoirement (tous inférieurs á 10^5 pour que les tests s'effectuent de manière relativement rapide).

On obtient les résultats suivants :

```
In [393]: proba_erreur_test_fermat(100)
Out[393]: 0.31
```

Ainsi la fonction **test_fermat** se trompe (i.e. pense qu'un nombre est premier alors qu'il est en réalité composé) environ 31 fois sur 100.

4 Exercice 4 : Test de Rabin et Miller

Nous voulons tester la probabilité d'erreur de notre fonction **test_miller_rabin** (implémentant l'algorithme de Miller-Rabin décrit dans l'énoncé) de la même manière que nous l'avons fait pour **test_fermat**. On obtient les résultats suivants :

```
In [893]: proba_erreur_test_miller_rabin(100,100)
Out[893]: 0.23
```

Le test de Rabin et Miller semble donc meilleur que le test de Fermat dont la probabilité d'erreur était d'environ 30%.

Enfin, nous utilisons la fonction **test_miller_rabin** pour écrire une fonction **gen_rsa** permettant de génerer des nombres pseudo-premiers.
En choisissant en entrée $t = 10$ on obtient, par exemple, le résultat suivant :

```
In [892]: gen_rsa(10)
Out[892]: 475978
```