MPSI 1 **Mardi 8 janvier 2019**

DS 2 IPT : ARITHMÉTIQUE ET INFORMATIQUE

Entrée chinoise :

Supposons par exemple que $m = \max(m,n)$. La méthode consiste simplement à chercher parmi les $a_k := a + km$ pour $k \in [\![0, n-1]\!]$ celui tel que $a_k \equiv b\,[n]$. Une fois ce représentant trouvé, on sait que le système équivaut à $\begin{cases} x \equiv a_k\,[m], \\ x \equiv a_k\,[n] \end{cases}$ ce qui équivaut encore (car $m \wedge n = 1$) à $x \equiv a_k\,[mn]$. Il suffit donc de renvoyer a_k (le return force la sortie de la fonction et donc de la boucle `for`).

```
def f(m,n,a,b):
    # echange des var. pour avoir
    # m=min(m,n)
    if n>m:
        m,n=n,m
        a,b=b,a
    # calcul et test des ak
    for k in range(1,n):
        ak=a+k*m
        if ak%n==b%n:
            return ak
```

Le programme fait bien au maximum $n = \min(m,n)$ tours de boucles et à chaque tour de boucle, il ne fait qu'un nombre constant d'opérations d'où la complexité en $O(\min(m,n))$.

1) L'algorithme des divisions successives :

a) Soit p le plus petit diviseur positif de m , différent de 1. Comme m n'est pas premier $m = pq$ avec $p \leq q$ donc $p^2 \leq pq$ donc $p^2 \leq m$ donc $p \leq \sqrt{m}$.

b)

```
def Testpremier(n):
    m=int(n**(1/2))
    for i in range(2,m+1):
        if n%i==0:
            return False,i,n//i
    return True
```

c)
```
def Decompose(n):
    m=int(n**(1/2))
    for i in range(2,m+1):
        if n%i==0:
            return i,n//i
    return True
```

d) Nombre de chiffres dans l'écriture décimale :

i) Si $x \in \mathbb{R}^{+*}$, s'écrit avec N chiffres en base 10, alors $10^{N-1} \leq x < 10^N$ donc $N-1 \leq \log_{10}(x) < N$ donc $N - 1 = \lfloor \log_{10} x \rfloor$ où encore $N = \lfloor \log_{10} x \rfloor + 1$.

ii) En posant $y = x/2$ la propriété à démontrer est équivalente à $\lfloor y \rfloor = \lfloor \dfrac{\lfloor 2y \rfloor}{2} \rfloor$ pour $y \geq 0$.
On note $y = \lfloor y \rfloor + r_y$ avec $r_y \in [0,1[$. Alors $2y = 2\lfloor y \rfloor + 2r_y$ et comme $2\lfloor y \rfloor$ est un entier, on en déduit que $\lfloor 2y \rfloor = 2\lfloor y \rfloor + \lfloor 2r_y \rfloor$ puis que :
$\dfrac{\lfloor 2y \rfloor}{2} = \lfloor y \rfloor + \dfrac{\lfloor 2r_y \rfloor}{2}$

Or $0 \leq r_y < 1$ donc $0 \leq 2r_y < 2$ donc $\lfloor 2r_y \rfloor \in \{0,1\}$ et $\dfrac{\lfloor 2r_y \rfloor}{2} \in \{0, 1/2\}$, donc $\lfloor y \rfloor = \lfloor \dfrac{\lfloor 2y \rfloor}{2} \rfloor$ c.q.f.d.

iii) Si m est un nombre ayant N chiffres en base dix, on sait que $N = \lfloor \log_{10}(m) \rfloor + 1$.
Et le nombre N' de chiffres de $\sqrt{m}$ vérifie $N' = \lfloor \log_{10} \sqrt{m} \rfloor + 1 = \lfloor \frac{1}{2} \log_{10}(m) \rfloor + 1$
Donc d'après la question qui précède : $N' = \lfloor \frac{\lfloor \log_{10}(m) \rfloor}{2} \rfloor + 1 = \lfloor \frac{N-1}{2} \rfloor + 1$

e) L'algorithme du a) ou b) sera le plus long à répondre si la sortie de boucle est la plus proche possible de $\sqrt{m}$. Donc les nombres non premiers à N chiffres les plus long à détecter seront ceux de la forme p^2 où p est un nombre premier à N' chiffres où N' est relié à N comme à la question précédente.

2.1.)

a)
```
def TestCarre(n):
    a=int(n**(1/2))
    return n==a**2
```

b) En abusant de fonctions puissantes de Python comme le `in` et le `sort`, on peut obtenir la liste sans répétition, et triée :

```
 L=[]
for i in range(100):
    a=i**2%100
    if a not in L:
            L.append(i**2%100)
L.sort()
```

La liste complète (pour information) (où j'ai rajouté des 0 devant les nombres à un chiffre) :
`[00, 01, 04, 09, 16, 21, 24, 25, 29, 36, 41, 44, 49, 56, 61, 64, 69, 76, 81, 84, 89, 96]`

c)
```
 def DecompFermat(m):
    xk=ceil(m**(1/2))
    k=0
    while TestCarre(xk**2-m)!=True:
        xk=xk+1
    yk=int(sqrt(xk**2-m))
    return (xk-yk,xk+yk)
```

d) **Remarque préliminaire :** Si m est impair, on peut toujours trouver un couple $(x, y) \in \mathbb{N}^2$ tel que $x^2 - y^2 = m$.

En effet pour la décomposition triviale, $m = 1 \times m$ on cherche $(x, y) \in \mathbb{N}^2$ tels que $\begin{cases} x - y = 1, \\ x + y = m \end{cases}$
on trouve donc $x = 1 + y$ et $2y + 1 = m$ donc comme m est impair, $y = (m-1)/2$ entier et $x = 1 + m$.
Donc on a bien un point à coordonnée entière sur $\mathcal{H}$.
Pourquoi la remarque prouve l'arrêt de l'algorithme :
La décomposition trouvée : si m n'est pas un carrée, c'est celle en $m = p.q$ avec p le plus grand diviseur de m tel que $p < \sqrt{m}$ et donc q le plus petit diviseur de m tel que $q > \sqrt{m}$.
Pour $m = 3^2.7 = 63$, on aura donc $p = 7$ et $q = 9$ de part et d'autre de $\sqrt{63}$ qui est proche de 8.

2.2)
a) On note $x = x_0 = \lceil \sqrt{m} \rceil$. Alors $x_{k+1}^2 = (x + (k+1))^2 = ((x+k)+1)^2 = x_k^2 + 2x_k + 1$.
Donc $x_{k+1}^2 - m = (x_k^2 - m) + 2x_k + 1$.
b)

```
def DecompFermat2(m):
    xk=ceil(m**(1/2))
    xk2=xk**2
    while TestCarre(xk2-m)!=True:
        xk2=xk2+2*xk+1
    yk=int(sqrt(xk2-m))
    return (xk-yk,xk+yk)
```

2.3)

Notons $x_0 = \lceil \sqrt{m} \rceil$. Lorsque l'algo. s'arrête en un $x_k = x_0 + k$ alors $(x_0+k)^2 - y_k^2 = m$ avec y_k entier et la décomposition obtenue est

$(x_0 + k - y_k)(x_0 + k + y_k) = m$.

Donc en notant $p = x_0 + k - y_k$ et $q = x_0 + k + y_k$ on trouve bien que $(p+q)/2 = x_0 + k$. Donc le nombre k d'étape de l'algorithme est exactement égal à $\mu - x_0 = \mu - \lceil \sqrt{m} \rceil$.

3.0)

3.1) a) $a_k = 2^{(k-1)! \times k} = a_{k-1}^k$. Donc à partir de $a_1 = 2$ on fait

b) Par déf. $g_k = pgcd((a_k \% m) - 1, m)$.

Notons $r = a_k \% m$. Par déf. de la division euclidienne $a_k = qm + r$ et donc $(a_k - 1) = qm + (r-1)$.

Par le lemme de base de l'algorithme d'Euclide, on sait alors que l'ensemble $\Delta(a_k - 1, m)$ des diviseurs communs à $a_k - 1$ et m, est le même que l'ensemble $\Delta(m, r-1)$ des diviseurs communs à m et $r-1$.

Démontrons ce lemme vu la question posée : on rappelle que si un nombre d divise deux nombres α et β alors d divise toutes les C.L à coeff. entiers $k\alpha + l\beta$.

Ici si d divise $a_k - 1$ et m alors d divise $a_k - 1 - qm$ donc d divise $r-1$. Réciproquement si d divise m et $r-1$ alors d divise $qm + (r-1)$ donc d divise $a_k - 1$. □

Pourquoi calculer plutôt pgcd$((a_k \% m) - 1, m)$? Parce qu'ainsi à chaque étape la taille des nombres en jeu ne dépasse pas m.

c) On va montrer mieux : pour chaque k tel que g_k et g_{k+1} sont définis, g_k divise g_{k+1}.

Par déf. $a_k = a_{k-1}^k$ donc $a_k - 1 = a_{k-1}^k - 1 = (a_{k-1} - 1)(a_{k-1}^{k-1} + \cdots + a_{k-1} + 1)$.

En particulier $(a_{k-1} - 1) | (a_k - 1)$.

Mais alors $g_{k-1} = \text{pgcd}(a_{k-1} - 1, m)$, qui est un diviseur de $a_{k-1} - 1$ est aussi un diviseur de $a_k - 1$. Donc g_{k-1} est un diviseur commun à $a_k - 1$ et à m ce qui entraîne que g_{k-1} divise le pgcd$(a_k - 1, m)$ autrement dit $g_{k-1} \mid g_k$.

d) Remarque : Bien sûr pour p premier impair, $p-1$ est pair, et donc il ne sera pas nécessaire d'aller jusqu'à $n = (p-1)$ pour que $n!$ soit divisible par $(p-1)$.

Venons-en à la question posée : par petit Théorème de Fermat, comme p est premier, $2^{p-1} - 1 = pk$ avec $k \in \mathbb{N}$.

Donc en notant $n! = (p-1)q$, $2^{n!} - 1 = (2^{p-1})^q - 1 = (2^{p-1} - 1)Q$ avec $Q \in \mathbb{N}$ par la même identité remarquable qu'au c).

Donc comme $p | 2^{p-1} - 1$ on en déduit bien que $p | (2^{n!} - 1)$.

Ainsi $p | a_k - 1$ et m donc $p | g_k$. □

3.2. a)

```
def Pollard(m):
    g=1 # valeur de g_1
    a=2 # valeur de a_1
    k=2 # valeur de k avec laquelle
    # on va commencer la boucle
    while g==1:
        a=a**k
        g=pgcd(a%m-1,m)
        k=k+1
    return g
```

b)

$a_1 \equiv 2[65]$ $\quad 1 \wedge 65 = 1,$
$a_2 = 2^2 \equiv 4\ [65]$ $\quad 3 \wedge 65 = 1,$
$a_3 = 2^6\ \equiv 64\,[65] \equiv -1\,[65]$ $\quad -1 \wedge 65 = 1,$
$a_3 = 2^{24} \equiv (-1)^4 \equiv 1\,[65]$ $\quad 0 \wedge 65 = 65.$ Echec : on n'a pas trouvé de diviseur non trivial.

c) Explication de l'échec pour $m = 65 = 5 \times 13$. Il se trouve qu'en même temps ici $5 - 1 = 4$ et $13 - 1 = 12$ divisent tous les deux $4! = 24$. Donc le petit théorème de Fermat donne que $a_4 \equiv 1\ [5]$ et $a_4 \equiv 1\,[13]$. On en déduit que $a_4 \equiv 1\ [65]$.

d) Le même échec se produira (notamment) si $m = pq$ avec p et q deux nombres premiers tels que $(p-1)$ et $(q-1)$ divisent *en même temps* l'exposant $n!$ pour la première fois ! Dans ce cas on aura $g_{n-1} = 1$ et $g_n = pq$.

3.3. a) Algorithme avec $a_1 = 3$ et $m = 65$.

$a_1 \equiv 3[65]$ $\quad 2 \wedge 65 = 1,$
$a_2 = 3^2 \equiv 9\ [65]$ $\quad 8 \wedge 65 = 1,$
$a_3 = 2^6\ \equiv 14\,[65]\,[65]$ $\quad 13 \wedge 65 = 13.$

L'algorithme va bien retourne le facteur premier 13 de 65.

b) Pourquoi l'algorithme réussit-il ? A priori le théorème de Fermat s'applique de la même manière à 2 et à 3 modulo 13 pour donner que $2^{12} \equiv 1\,[13]$ et $3^{12} \equiv 1\,[13]$.

Mais la différence entre les deux cas de 2 et de 3 vient ici du fait que 12 est le plus petit exposant non nul a tel que $2^a \equiv 1\,[13]$ (on dira que $\overline{2}$ est d'ordre 12 dans $(\mathbb{Z}/13\mathbb{Z}^*, \times)$) alors que pour 3, on a déjà $3^3 \equiv 1\,[13]$ (on dira que $\overline{3}$ est d'ordre 3 dans $(\mathbb{Z}/13\mathbb{Z}^*, \times)$.).

Ainsi en changeant de « témoins de Fermat », dans l'algorithme, on peut diminuer les ordre.

A faire. la maison : essayer de le faire pour le cas très défavorable de $m = 1891$.