

Devoir Surveillé n°2

Durée de l'épreuve : 2 heures
La calculatrice est interdite

Le devoir est composé d'un exercice et d'un problème.

Pour l'écriture des programmes, il sera important de :

- bien choisir le nom des variables et des fonctions utilisées,
- bien documenter les algorithmes,

afin de bien se faire comprendre du correcteur. Cela sera un critère important de la notation.

BON COURAGE

Exercice - Cryptarithme

Un cryptarithme est une devinette calculatoire du type :

$$\text{MORE} + \text{SEND} = \text{MONEY}$$

où chaque lettre désigne un chiffre (entre 0 et 9) de sorte que l'opération ainsi créée soit exacte. Ce problème, donné en exemple, admet une unique solution : $1085 + 9567 = 10652$ (donc $M = 1$, $O = 0$, $Y = 2$, etc.)

Créer un programme qui résout le cryptarithme suivant :

$$\text{UN} + \text{UN} + \text{NEUF} = \text{ONZE} \quad (E)$$

Ce programme étudiera tous les cas possibles. Il pourra donc exploiter les sous-programmes suivants qui faudrait alors rédiger ... :

- *Essai* : étant donné une liste de 6 chiffres, ce programme teste si en remplaçant les lettres U, N, E, F, O, Z par ces chiffres, l'équation (E) est vérifiée
- *Liste_6* : algorithme qui teste si (au moins) deux chiffres sont identiques dans une liste de 6 chiffres (en faisant par exemple le produit de toutes les différences possibles des nombres de la liste pris deux à deux.)

Problème - Nombres pseudo-premiers et cryptage RSA

A. Nombres pseudo-premiers

On rappelle qu'un nombre premier p est un nombre entier positif qui n'admet que 1 et lui-même comme diviseur.

Un nombre qui n'est pas premier est dit nombre composé.

Un nombre n est donné en argument. On cherche à savoir, si ce nombre est premier.

1. Voici un extrait de codage en Python de l'algorithme d'Erathostène.

```
1 def crible(n):  
2     """  
3     Pour un entier n (>=2), cette fonction renvoie la liste  
4     des nombres premiers inferieurs ou egaux a n. Cette liste  
5     est construite avec la methode du crible d'Erathosthene  
6     """  
7     liste = list(range(2,n+1))  
8     premiers = []  
9     while liste != []:  
10        p = liste[0]  
11        premiers.append(p)  
12        for n in liste:  
13            if n%p == 0:  
14                liste.remove(n)  
15    return(premiers)
```

- (a) Expliquer le rôle des lignes 7 et 11
- (b) Justifier qu'un tel programme se termine bien en un temps fini ?

- (c) On note $T(n)$ le temps mis par un tel programme pour afficher le crible d'Erathostène pour les nombres entiers inférieurs à n .

On constate expérimentalement que pour tout $n \in \mathbb{N}$, $T(2n) = 4T(n)$, on considère donc que pour tout x réel positif, $T(2x) = 4T(x)$.

Montrer alors que pour tout $x > 0$, $\frac{T(x)}{x^2} = \frac{T(\frac{x}{2})}{(\frac{x}{2})^2}$, puis que $\frac{T(x)}{x^2} = \frac{T(\frac{x}{2^n})}{\frac{x^2}{2^{2n}}}$

en déduire, en admettant qu'il existe $A = \lim_{y \rightarrow 0} \frac{T(y)}{y^2}$, que : $\forall x > 0, T(x) = A \times x^2$.

- (d) Ce résultat expérimental : $T(n) = A \times n^2$, vous semble-t-il justifié ?

Expliquez rapidement pourquoi.

La méthode précédente, exhaustive, n'est pas très efficace.

2. On va plutôt essayer d'exploiter le (petit) théorème de Fermat :

Soit n un nombre premier. Alors pour tout $a < n$, $a^{n-1} \equiv 1[n]$

Donner la contraposée de la proposition précédente.

On appelle témoins de Fermat de n les nombres $a \in \llbracket 2, n \rrbracket$ tel que $a^{n-1} \not\equiv 1[n]$.

Avantage : il existe fréquemment de très petits témoins de Fermat. : $a = 2$ ou $a = 3$ suffit à détecter la majorité des nombres premiers.

3. Ecrire un programme qui teste si a est un témoin de Fermat du nombre n .
4. On dit qu'un nombre n est pseudo-premier, si 2 et 3 ne sont pas des témoins de Fermat de n .
 - (a) Ecrire un programme qui teste si un nombre n donné est un nombre pseudo-premier.
 - (b) On note $S(n)$ le temps mis par un tel programme pour évaluer si n est pseudo-premier. Expliquer pourquoi on peut affirmer qu'il existe B tel que $S(n) = B \times n$.
 - (c) Que pensez-vous de l'intérêt de la recherche des nombres pseudo-premiers, pour n grand ?

Malheureusement, il existe des nombres non premiers p et sans témoin de Fermat, c'est-à-dire qu'ils sont également pseudo-premiers. On appelle ces nombres, les nombres de Carmichael.

Ce plus petit nombre est $561 = 3 \times 11 \times 17$.

On sait maintenant qu'il existe une infinité « très rare » de tel nombre

B. RSA

Le principe de la cryptographie à clés publiques, de messages entre K personnes est le suivant (*A lire très rapidement le jour du DS*).

On note $\mathcal{M}$ l'ensemble des messages et chaque interlocuteur $i \in \llbracket 1, K \rrbracket$ choisit deux applications réciproques l'une de l'autre :

$$c_i : \mathcal{M} \rightarrow \mathcal{M}, \quad d_i : \mathcal{M} \rightarrow \mathcal{M}$$

L'application c_i est l'application de codage (ou chiffrement) de l'interlocuteur i , l'application d_i son application de décodage (ou déchiffrement).

Les c_i sont connus de tous, et l'application $d_i = (c_i^{-1})$ n'est connue que de son propriétaire i .

Pour envoyer le message $m \in \mathcal{M}$ à l'interlocuteur j , l'interlocuteur le chiffre ou le code en lui appliquant la fonction $c_j \circ d_i$ (il connaît sa fonction d_i et toutes les fonctions c_j) et envoie donc $m' = c_j(d_i(m))$.

A la réception du message chiffré m' , l'interlocuteur applique la fonction $c_i \circ d_j$ à m' et retrouve ainsi le bon message m .

Pour que la cryptographie à clés publiques soit efficace, il faut que le calcul (algorithmique) $c_i(m)$ soit rapide, mais que trouver m à partir de c_i est difficile (casser c_i en d_i).

RSA

En 1978, R. Rivest, A. Shamir et L. Adleman proposent un mécanisme adapté, exploitant le calcul des congruences.

Le principe est le suivant : l'interlocuteur i considère deux nombres premiers p_i et q_i et annonce publiquement le nombre $n_i = p_i \times q_i$, ainsi qu'un nombre r_i , premier avec le nombre $n'_i = (p_i - 1)(q_i - 1)$ (gardé secret).

L'application de chiffrement est alors $c_i : m \mapsto m^{r_i} [n_i]$.

Pour déchiffrer le message, l'interlocuteur i exploite le nombre s_i tel que $r_i \times s_i \equiv 1 [n'_i]$.

Plus précisément, $d_i : m' \mapsto (m')^{s_i} [n_i]$.

A l'heure actuelle, il n'existe pas d'algorithme « rapide et simple » pour trouver s_i connaissant r_i et n_i (pour p_i et q_i de grands nombres premiers).

1. Pourquoi est-il suffisant de considérer $\mathcal{M}$ comme un ensemble d'entiers pour coder tout type de message ?
2. Ecrire un programme qui fait l'opération c_i , les nombre m , r_i et n_i sont donnés en argument.
3. Pour l'interlocuteur i , connaissant p_i , q_i , il faut trouver un nombre r_i premiers avec $n'_i = (p_i - 1)(q_i - 1)$ et un nombre s_i tel que $r_i \times s_i \equiv 1 [n'_i]$.
On remarque que $r_i \times s_i \equiv 1 [n'_i]$ si et seulement si $\exists h_i \in \mathbb{Z}$ tel que $r_i s_i + h_i n'_i = 1$.
Compléter les lignes 17 et 20 du code suivant pour que le programme renvoie un couple (r_i, s_i) satisfaisant.

```
1 def Bezout(a,b):
2     """
3     Renvoie sous forme de liste (triplet),
4     1. le pgcd de a et b
5     2. un nombre u
6     3. un nombre v
7     tels que pgcd(a,b)=au+bv
8     """
9     (...)
10
11 from random import randint
12
13 def choix(p,q):
14     """
15     Choisit r aléatoirement, premier avec n'=(p-1)(q-1)
16     Il est renvoyé avec s tel que rs=1[n']
17     """
18     n2=(p-1)*(q-1)
19     r=randint(2,n2-1)
20     L=Bezout(... , ...)
21     while L[0]!=1:
22         r=randint(2,n2-1)
23         L=Bezout(... , ...)
24     s=L[1]%n2
25     return(r,s)
```

On suppose : $\text{Bezout}(a,b) \mapsto [a \wedge b, u, v]$ avec $a \wedge b = u \times a + v \times b$

On ne demande pas de programmer *Bezout*, cela a déjà été fait.

4. Dans la pratique, on ne choisit pas p_i et q_i premiers, mais seulement pseudo-premiers (voir remarque plus loin).
Ecrire un algorithme qui choisit aléatoirement deux nombres pseudo-premiers de 4 chiffres.
On ne cherchera pas à montrer la terminaison de l'algorithme.
Pour 4 chiffres, Python effectue les calculs en moins d'une minute.
5. Comment exploiter vos algorithmes pour échanger secrètement des messages avec le reste de la classe ?

Remarques sur RSA

L'enjeu consiste donc à trouver et garder secret deux grands nombres entiers p_i et q_i . La factorisation de n_i (connu) en $p_i \times q_i$ est un problème complexe à l'heure actuel...

Mais, tout n'est pas si simple, nous savons toutes la difficulté à trouver de grands nombres p_i et q_i . Dans la pratique, nous exploitons avec RSA des nombres pseudo-premiers. Il s'agit de nombre qui passent positivement à un certain nombre de test comme le test des témoins de Fermat vu en première partie.

Dans la pratique cela marche quasiment à tous les coups. Il faut juste s'assurer avant d'envoyer m que $c_i \circ d_i(m) = m$, ce qui est heureusement, bien souvent le cas...