Nombres premiers : tests de primalité, cryptographie

N'oubliez pas de sauvegarder vos sessions (format .xws) sur la plate-forme, ainsi que les programmes (format .cxx).
ATTENTION : Il est obligatoire de rendre certains de ces exercices avant la fin du TP.

Tests de primalité

La recherche de grands nombres premiers étant devenue cruciale, en particulier pour la cryptographie, il est important de disposer d'algorithmes performants testant la primalité de nombres entiers.

Exercice 1. *Test de primalité déterministe naïf*
L'algorithme le plus simple consiste à tester s'il existe un nombre entier supérieur ou égal à 2 et strictement inférieur à N divisant N. Si oui, alors N n'est pas premier. La complexité (vitesse de calcul) de cet algorithme est donc directement liée à la taille de l'entier N à tester, elle est donc très lente, de type exponentielle.
Une première amélioration de cet algorithme consiste alors à ne tester que les entiers inférieurs ou égaux à $\sqrt{N}$; sauriez-vous expliquer pourquoi ?

Implémentez cet algorithme :

```
Programme test_naif(N)
 Variables:  j (entier), t (booléen)
 t:=0
 j:=2
 Tant que t=0 ET j<= partie_entière(racine(N))
  Si j divise N alors t:=1
  FinSi
  j:=j+1
 FinTantque
 Si t=0 alors afficher "N est premier"
          sinon afficher "N n'est pas premier"
 FinSi
```

Pour aller plus loin : Vous pouvez ensuite améliorer votre programme pour qu'il soit encore plus efficace ; par exemple, il peut repérer les nombres pairs ou impairs, et ensuite ne tester que des diviseurs potentiels impairs de N. On pourrait faire de même avec les nombres multiples de 3.

Exercice 2. *Test de primalité probabiliste de Fermat*

Ce test est basé sur le théorème suivant, appelé petit théorème de Fermat :

$$\text{Si } N \text{ est premier, alors pour tout } a \text{ premier avec } N,\ a^{N-1} \equiv 1 \mod N.$$

Ainsi, on dispose d'un test de primalité probabiliste :
On choisit un entier $2 \leqslant a \leqslant N-1$ et on calcule $a^{N-1} \mod N$.
Si on obtient un nombre différent de 1, N est à coup sûr non premier, et si on obtient 1 alors N est probablement premier.

a. Implémentons une première version :

```
Programme test_fermat(a,N)
 Variables: b
 b:=a^(N-1) modulo N
 Si b=1 alors Afficher("N est probablement premier")
 Sinon Afficher("N n'est pas premier")
 FinSi
```

Le tester avec $a = 2$, $a = 3$ pour quelques entiers. Vérifiez les résultats obtenus avec la fonction `is_prime`. Comparez les temps de calculs avec l'algorithme naïf de l'exercice 1. Nous avons gagné en vitesse d'exécution par rapport à l'algorithme naïf, ici nous avons un temps d'exécution de type polynomial.

b. Modifiez votre programme pour qu'il choisisse a au hasard (entre 2 et $N - 1$). On pourra utiliser la fonction `hasard` de Xcas.

c. Mise en défaut !
Testez vos programmes précédent avec le nombre 252601. Ce nombre est-il premier ? Que renvoie-t-il avec vos différents algorithmes ?
Ce nombre est un nombre dit de Carmichaël, qui met en défaut le test de Fermat. En effet, si N est de Carmichaël, alors pour tout entier a premier avec N, $a^{N-1} \equiv 1 \mod N$. Il a été prouvé en 1994 qu'il existait une infinité de nombres de Carmichaël. Heureusement ils sont rares : il n' y en a que 105212 inférieurs à 10^{15}. (d'après Wikipédia)

Pour aller plus loin : Implémentez un algorithme qui recherche le premier nombre de Carmichaël.

Exercice 3. *Test de primalité déterministe pour les nombres de Mersenne*

Actuellement, les recherches de grands nombres premiers se font à l'aide des nombres de Mersenne, qui sont des entiers de la forme $2^n - 1$.
Découvert le 25 janvier 2013, le plus grand nombre premier connu est le nombre premier de Mersenne $2^{57885161}$–1, qui comporte 17 425 170 chiffres en écriture décimale. On le doit à l'équipe de Curtis Cooper, à l'université du Central Missouri, dans le cadre de la grande chasse aux nombres premiers de Mersenne (GIMPS). Écrits les uns à la suite des autres, ses chiffres occuperaient plus de 4 000 pages en police Times New Roman taille 12. (d'après Wikipédia)
Il est aisé de démontrer le résultat suivant : Si $2^n - 1$ est premier alors n est premier. Malheureusement, la réciproque est fausse (tester avec $n = 11$). Malgré tout, cela signifie que l'on pourra chercher de grands nombres premiers sous la forme $M_p = 2^p - 1$, avec p premier.

Il existe alors un algorithme, appelé test déterministe de Lucas-Lehmer, permettant de déterminer si un nombre de Mersenne M_p est premier.
Il est basé sur la propriété suivante, où p est impair :

M_p est premier si et seulement si M_p divise S_{p-1} avec $S_1 = 4$ et pour $n > 1$, $S_{n+1} = S_n^2 - 2$.

Implémentez ce test. Il sera avantageux de faire tous les calculs concernant la suite S_n directement modulo M_p.
Le tester avec M_{59}, M_{107}, M_{607}, M_{3001}, M_{3217}.

Exercice 4. *Test de primalité probabiliste de Solovay-Strassen*

Ce test de primalité est une amélioration du test probabiliste de Fermat, élaboré en 1976 par les mathématiciens Solovay et Strassen.
Ce test est basé sur le résultat suivant, où $\left(\frac{a}{b}\right)$ désigne le symbole de Jacobi :

$$\text{Si } N \text{ est premier impair, alors si } a \text{ premier avec } N,\ \left(\frac{a}{N}\right) \equiv a^{(N-1)/2} \mod N.$$

Ainsi, on dispose d'un test de primalité probabiliste (car la réciproque du résultat précédent n'est pas réalisée), où N est un entier naturel impair :
On choisit aléatoirement un entier $1 \leqslant a \leqslant N-1$ et on teste si $\left(\frac{a}{N}\right) \equiv a^{(N-1)/2} \mod N$. S'il y a égalité, N est probablement premier, sinon, N est à coup sûr non premier.
L'intérêt de ce test par rapport au test de Fermat est qu'il n'existe pas l'équivalent des nombres de Carmichaël. Autrement dit, et plus précisément, pour un entier N composé, et si a est premier avec N il y a au moins une chance sur deux que $\left(\frac{a}{N}\right) \not\equiv a^{(N-1)/2} \mod N$.
On peut proposer alors l'algorithme suivant, pour lequel on peut montrer que la probabilité qu'il détecte un nombre premier est égale à $1 - \frac{1}{2^k}$:

```
Programme test_SoSt(k,N)  // k est un entier >=1 et N un entier impair à tester
 Variables: a,j,t
 t:=0   // variable test qui reste à 0 si N est premier
 j:=1
 Tant que t=0 et j<=k
 On choisit a au hasard tel que  1<=a<=N-1 et a premier avec N
 // attention, on doit être sûr que a est premier avec N
 Si jacobi(a,N) n'est pas égal à a^(N-1)/2 modulo N alors t:=1 FinSi
 j:=j+1
 FinTantque
 Si t=1 alors Afficher("N n'est pas premier")
 Sinon Afficher("N est probablement premier")
 FinSi
```

Le symbole de Jacobi sur Xcas est obtenu par la commande `jacobi_symbol`. Cet algorithme est de complexité polynômiale.

Cryptographie

Exercice 5. *Système RSA*

Le principe du système RSA (mis au point en 1977 par Rivest, Shamir et Adelman), repose sur le fait qu'il est très difficile de factoriser un grand nombre entier, alors qu'au contraire il est plus aisé de trouver de grands nombres premiers.
Le système RSA est ainsi un système de cryptographie à clé publique, permettant le chiffrement d'un message, tandis que la clé de déchiffrement reste secrète et résiste aux attaques malveillantes.

Voici plus en détail son fonctionnement :
- Données : La clé publique consiste en la donnée d'un nombre N qui est en fait produit de deux nombres premiers p et q (N=pq), et d'un entier naturel c qui est premier avec $n = (p-1)(q-1)$. Attention, les nombres premiers p et q doivent rester secrets.
- Etape de chiffrement : Si le message à envoyer est un entier $a < N$, le chiffrement consiste à le transformer en un entier b défini par $b \equiv a^c \mod N$, avec $0 \leqslant b < N$. On suppose ici que a est premier avec N.
- Etape de déchiffrement : L'entier b reçu est déchiffré en utilisant la relation : $a \equiv b^d \mod N$, où d est l'entier tel que $cd \equiv 1 \mod n$, avec $0 \leqslant d < n$. d n'est autre qu'un inverse de c modulo n, il est donc facilement calculable par l'algorithme d'Euclide étendu, si on connaît n (nombre resté secret, seulement connu de la personne autorisée à déchiffrer).

Remarque : montrez en utilisant le petit théorème de Fermat que l'étape de déchiffrement renvoie bien le message initial a.

a. Premiers codages.

Implémentez un premier algorithme (ou une fonction) de chiffrement, ayant pour entrée la clé publique, c'est-à-dire le couple (N, c), et un entier a à coder. Le résultat doit être l'entier b.
Implémentez ensuite l'algorithme (ou fonction) de déchiffrement, dont les entrées sont la clé secrète de déchiffrement, c'est-à-dire le couple (N, d), et un entier b à décoder. Le résultat doit être l'entier a.
Vérifiez que vos algorithmes fonctionnent bien en choisissant p, q deux nombres premiers, $N = pq$, un entier c premier avec $n = (p-1)(q-1)$, et d l'inverse de c modulo n.

b. Pour coder des messages textuels.
Nous allons utiliser l'écriture de nombres en base 26 pour coder des messages textuels. Tout d'abord, associons à toute lettre de l'alphabet son rang dans l'alphabet moins 1 ; ainsi, le A vaut 0, le B vaut 1,..., le Z vaut 25 (cf tableau ci-après).
A un mot de n lettres $L_1...L_n$, on peut alors associer un nombre entier $a_1 26^0 + a_2 26^1 + ... + a_n 26^{n-1}$, où les a_i sont les entiers compris entre 0 et 25 associés à la lettre L_i. Réciproquement, à tout entier de la forme $a_1 26^0 + a_2 26^1 + ... + a_n 26^{n-1}$ on peut associer un mot de n lettres $L_1...L_n$.
Le nombre $a = a_1 26^0 + a_2 26^1 + ... + a_n 26^{n-1}$ est alors un entier dont l'écriture en base 26 est $\overline{a_1 \cdots a_n}$. Nous pouvons alors utiliser un codage RSA pour chiffrer et déchiffrer notre entier a, à condition de choisir N assez grand ($N \geqslant 26^n$, et si possible aussi $p, q > 26^n$, pour que tous les entiers à coder soient premiers avec N).
Un exemple, avec $n = 3$. Le mot ABC est associé au nombre $a = 0 \cdot 26^0 + 1 \cdot 26 + 2 \cdot 26^2 = 1378$. On peut le vérifier avec Xcas, en tapant la commande `convertir([0,1,2],base,26)`. Si on code a avec la clé ($N = 19633, c = 5$), où $p = 677$, $q = 29$, on obtient $b = 17352$. Grâce à la commande `convertir(17352,base,26)` on obtient $b = 10 \cdot 26^0 + 17 \cdot 26^1 + 25 \cdot 26^2$, soit le mot KRZ. La clé de déchiffrement est alors ($N = 19633, d = 11357$). Vérifiez tout cela en utilisant vos algorithmes/fonctions de a.
Implémentez de nouveaux algorithmes (fonctions) de chiffrement permettant par cette méthode de (dé)coder des mots de n lettres.

Pour aller plus loin : On pourra essayer de convertir directement des mots en nombres en trouvant des fonctionnalités adéquates de Xcas.

Tableau de conversion :

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Exercice 6. *Chiffrement de Hill*

Voici le principe de ce chiffrement, qui est un système de cryptage symétrique, à clé non publique, et relativement facile à casser par des méthodes fréquentistes. C'est une généralisation de la méthode par chiffrement affine.
On se donne un texte que l'on transforme en série de nombres entiers compris entre 0 et 25 (cf tableau ci-dessus). Ensuite, on regroupe chacun de ces nombres par paires successives (en rajoutant éventuellement un nombre à la fin). Pour une paire (n_1, n_2) de tels nombres, on utilise la clé de codage A qui est une matrice 2×2 à coefficients dans $\mathbb{Z}/26\mathbb{Z}$, ie. $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ où les entiers a, b, c, d sont des entiers compris entre 0 et 25.
Le codage consiste alors à effectuer le produit $(n_1, n_2) * A = (m_1, m_2) \mod 26$ (c'est-à-dire que l'on effectue tous les calculs dans $\mathbb{Z}/26\mathbb{Z}$). En regroupant toutes les paires (m_1, m_2) et en revenant aux lettres, on obtient le texte codé.
Le décodage s'effectue en multipliant par la matrice inverse de A : $(n_1, n_2) = (m_1, m_2) * A^{-1} \mod 26$. Cela suppose que l'on a au départ choisi une matrice dont le déterminant est inversible modulo 26.
Comme il est aisé de décoder en connaissant A, il est indispensable que la clé A soit gardée secrète.

Un exemple : Avec $A = \begin{pmatrix} 3 & 5 \\ 6 & 17 \end{pmatrix}$, le mot ABCD devient GRYJ. Le vérifier. Que vaut A^{-1} ?

Utilisez Xcas pour expérimenter ou implémenter cette méthode de chiffrement. On pourra ensuite travailler avec des matrices de taille $p \times p$ (et donc regrouper les lettres par blocs de taille p).