Licence STS Mention MATH L3 - ATN
Arithmétique

1 Entiers rationnels

L'anneau $\mathbb{Z}$ des entiers est connu :

```
sage: ZZ
Integer Ring
sage: ZZ == IntegerRing()
True
sage: -5 in ZZ
True
sage: x = 10^33
sage: b =1479
sage: c = x.digits(b)
sage: c
[1081, 720, 612, 227, 1085, 1257, 645, 1120, 832, 1430, 19]
sage: add(c[k]*b^k for k in range(len(c))) == x
True
```

la division euclidienne, les pgcd et ppcm et la formule de Bezout s'obtiennent comme suit :

```
sage: a,b = 62735572, 49362
sage: a,b
(62735572, 49362)
sage: r = a.mod(b)
sage: r
45832
sage: q,r = a.quo_rem(b)
sage: q,r
(1270, 45832)
sage: a == b*q+r
True
sage: gcd(a,b)
2
sage: lcm(a,b)
1548376652532
sage: d,u,v = xgcd(a,b)
sage: d,u,v
(2, 5957, -7570921)
sage: d == a*u + v*b
True
```

On peut obtenir aussi les diviseurs d'un entier, les entiers premiers à un entier donnés et la fonction φ d'Euler :

```
sage: b.divisors()
[1, 2, 3, 6, 19, 38, 57, 114, 433, 866, 1299, 2598, 8227, 16454, 24681, 49362]
sage: lc = 250.coprime_integers(250)
sage: len(lc) == euler_phi(250)
```

Le test de primalité, et la factorisation d'un entier :les diviseurs premiers et la factorisation d'un entier sont fournis par :

```
sage: Primes()
Set of all prime numbers: 2, 3, 5, 7, ...
sage: p = 10263959282974110577205419657399167590071656780803806680334193352179071130777 9
          8033419335217907113077 79
sage: p
10263959282974110577205419657399167590071656780803806680334193352179071130777 9
sage: p.is_prime()
True
sage: lp = [p for p in range(2,100) if ZZ(p).is_prime()]
sage: len(lp) == prime_pi(100)
True
sage: next_prime(10^5)
100003
sage: fb = factor(b)
sage: fb
2 * 3 * 19 * 433
sage: list(fb)
[(2, 1), (3, 1), (19, 1), (433, 1)]
sage: b == expand(fb)
True
```

Le corps $\mathbb{Q}$ des nombres rationnels est pré-défini :

```
sage: QQ
Rational Field
sage: QQ == RationalField()
True
sage: QQ == FractionField(ZZ)
True
sage: r = 6/4
sage: r,r.numerator(),r.denominator()
(3/2, 3, 2)
sage: r in QQ
True
```

2 Entiers modulaires

Les anneaux $\mathbb{Z}/\mathbb{Z}n$ sont prédéfinis dans SAGE :

```
sage: R = IntegerModRing(23)
sage: R
Ring of integers modulo 23
sage: R.characteristic()
23
sage: R.order()
23
sage: R.is_field()
True
```

On peut déterminer des générateurs du groupe des éléments inversibles :

```
sage: R.multiplicative_generator()
5
sage: R.unit_gens()
[5]
sage: S = IntegerModRing(21)
sage: S
Ring of integers modulo 21
sage: S.is_integral_domain()
False
sage: S.multiplicative_generator()
Traceback (most recent call last):
...
ValueError: multiplicative group of this ring is not cyclic
sage: S.unit_gens()
[8, 10]
```

Soit $R = \mathbb{Z}/\mathbb{Z}n$; la classe $\overline{k}$ *modulo* n d'un entier rationnel k est obtenue en SAGE par la *coercition* `R(k)` ou par la fonction `Mod` :

```
sage: R = IntegerModRing(1048)
sage: R
Ring of integers modulo 1048
sage: a = R(14567)
sage: a
943
sage: a in R
True
sage: Mod(14567,1048) in R
True
sage: b = R(2)
sage: b
2
sage: a*b
838
sage: a^(-1)
519
```

On peut résoudre des équations modulaires :

```
sage: var('x')
sage: solve_mod([x^2 == 1],77)
[(1,), (43,), (34,), (76,)]
```

On peut résoudre un système de congruences $\begin{cases} x \equiv a \bmod m \\ x \equiv b \bmod n \end{cases}$ dans lequel les *modules* m et n sont premiers entre eux ; la solution x étant déterminée modulo mn.

```
sage: a = Mod(3,5)
sage: b = Mod(2,7)
sage: a.crt(b)
23
sage: crt(3,2,5,7)
-12
sage: Mod(-12,35)
23
```

3 Exercices

Exercice 1.
Pour $a = 1105$, $b = 208$ calculer le ppcm et le pgcd d de a et b. Déterminer des entiers $x, y \in \mathbb{Z}$ tels que la formule de Bezout $d = ax + by$ soit vérifiée.

Exercice 2.
Former la liste L des 15 premiers *entiers premiers* p tels que le nombre de Mersenne $M_p = 2^p - 1$ soit premier.
Vérifier que pour tout $p \in L$ le nombre $P_p = 2^{p-1}M_p$ est *parfait* (*ie.* égal à la somme de ses diviseurs propres.

Exercice 3.
Pour tout entier $n \geq 0$ on pose :

$$p_n = 2 + n[\frac{1}{1 + \sum_{k=2}^{n+1}[(n+2)/k - [(n+1)/k]]}]$$

où [] désigne la *partie entière*.

1. Former la liste L dont les éléments sont les p_n pour $0 \leq n \leq 100$.
   (Indication : la méthode `floor` permet de calculer la partie entière d'un nombre)
2. Former la liste P obtenue à partir de L en supprimant tous les éléments égaux à 2.
3. Soit r le nombre d'éléments de P ; vérifier que P est la liste des r premiers *entiers premiers* impairs.
   (Indication : vous pouvez utiliser un schéma de compréhension du type `[p for p ... if ...]`)

Exercice 4.
Le théorème de *Dirichlet* affirme, qu'une progression arithmétique dont le terme initial a et la raison q sont premiers entre eux contient une infinité de nombres premiers.
Ecrire en PYTHON une fonction `Dirichlet` qui étant donnés deux entiers a et q premiers entre eux et un entier N détermine la liste des N premiers termes de la progression arithmétique de terme initial a et de raison q qui sont premiers.

Exercice 5.
La *série de Farey* $\mathfrak{F}_n$ d'ordre n est la liste de tous les nombres rationnels de l'intervalle $[0, 1]$ dont le dénominateur est $\leq n$. On a $\mathfrak{F}_1 = \{0, 1\}$ et $\mathfrak{F}_n$ s'obtient à partir de $\mathfrak{F}_{n-1}$ en intercalant entre

deux nombres consécutifs $\frac{a}{b}$ et $\frac{c}{d}$ de $\mathfrak{F}_{n-1}$, le nombre rationnel $\frac{a+c}{b+d}$ lorsque l'on a $b+d \leq n$.
Ecrire en PYTHON une fonction *récursive* permettant de contruire $\mathfrak{F}_n$ et calculer $\mathfrak{F}_{10}$.

Exercice 6.
Résoudre chacun des systèmes suivants de congruences :

1. $\begin{cases} x \equiv 6 \bmod 17 \\ x \equiv 5 \bmod 11 \end{cases}$
2. $\begin{cases} x \equiv 3 \bmod 9 \\ 3x \equiv 10 \bmod 17 \end{cases}$

Exercice 7. On prend $p = 7$ et $n = 6$.

1. Combien le groupe $(\mathbb{Z}/p^n\mathbb{Z})^*$ contient-il d'éléments ?
2. Quel est l'ordre de $\overline{1+p}$ dans $(\mathbb{Z}/p^n\mathbb{Z})^*$? (on pourra utiliser une *boucle* puis utiliser la fonction SAGE `multiplicative_order()`).
3. Vérifier que le groupe $(\mathbb{Z}/p^n\mathbb{Z})^*$ est cyclique et trouver un générateur.

Problème 8. La méthode RSA
Le but est de transmettre un message confidentiel en toute sécurité. La méthode RSA est à *clé publique* c'est à dire que la méthode de codage est connue de tous (donc tout le monde peut fabriquer un cryptogramme). Mais seul le récepteur connaît le code de déchiffrage, et ce dernier ne peut pas être retrouvé à l'aide des informations rendues publiques. (cela repose sur le fait que les algorithmes connus de décomposition d'un entier en facteurs premiers nécessitent des temps de calculs importants pour des entiers dont les facteurs premiers sont très grands, de sorte que la connaissance d'un entier n n'implique pas nécessairement la connaissance immédiate de ses facteurs premiers.)

1. (a) On fixe un alphabet dont les caractères, numérotés de 0 à $l-1$ serviront à l'écriture des messages à transmettre. Cet alphabet est public par exemple :

   $'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.,;:\ '$

   Ici on a $l = 57$ caractères.

   (b) On fixe un entier $n = 436574584780292939766696226358147293030163397 91$; cet entier est public et est produit de deux grands entiers premiers distincts p et q gardés secrets.

   (c) On fixe un entier $c = 3$ (qui est lui aussi est public) tel que l'application :

   $$\begin{array}{rcl} C : \mathbb{Z}/\mathbb{Z}n & \longrightarrow & \mathbb{Z}/\mathbb{Z}n \\ x & \longrightarrow & x^c \end{array}$$

   soit bijective.

2. Ecrire en PYTHON une fonction `crypte` permettant de fabriquer un crytogramme `ctg` à partir d'un message `msg` écrit en n'utilisant uniquement les caractères de l'alphabet donné. Pour cela on procède de la façon suivante :

   (a) On remplace chaque lettre du message `msg` par son numéro dans l'alphabet ; on obtient ainsi une liste L d'entiers compris entre 0 et $l-1$.

   (b) On interprète cette liste L comme l'écriture d'un entier N en base l

(c) On écrit ce nombre N en base n. On obtient ainsi une liste M d'entiers compris entre 0 et $n-1$:

(d) On élève chaque élément de cette liste à la puissance c modulo n. On obtient ainsi une liste d'entiers compris entre 0 et $n-1$; c'est le cryptogramme `ctg` que l'on envoie au destinataire.

3. Seul le destinataire, qui connaît la *clé de décodage*

$$d = 29104972318686195959201875715334500356126826395$$

peut décoder ce message. L'application

$$\begin{array}{rcl} C^{-1} : \mathbb{Z}/\mathbb{Z}n & \longrightarrow & \mathbb{Z}/\mathbb{Z}n \\ y & \longrightarrow & y^d \end{array}$$

est la bijection réciproque de la bijection C. Ecrire en PYTHON une fonction `decrypte` permettant de reconstituer le message `msg` à partir du crytogramme `ctg` de la manière suivante :

(a) Le message codé `ctg` est une liste d'entiers compris entre 0 et $n-1$.

(b) Chaque élément de cette liste est élevé à la puissance d modulo n.

(c) On interprète cette liste comme l'écriture d'un entier N en base n.

(d) On écrit ce nombre N en base l. On obtient ainsi une liste L d'entiers compris entre 0 et $l-1$:

(e) On remplace chacun des ces numéros par la lettre correspondante de l'alphabet et le message d'origine `msg` est reconstitué.

4. On a $n = pq$ où p et q sont des entiers premiers distincts.

(a) Montrer que si c est premier avec $\varphi(n) = (p-1)(q-1)$ l'application C est bijective et que d est l'inverse de c *modulo* $\varphi(n)$

(b) Pourquoi connaissant n et c, le calcul de d revient-il à factoriser n ?

(c) Dans le problème on a pris $p = 378668090616600572642192533397$ et $q = 2^{60} - 173$; l'entier $n = pq$ se factorise immédiatement avec SAGE. Construire un exemple plus *réaliste*.