TP Maple 7 | Arithmétique dans $\mathbb{Z}$

L'Arithmétique (des entiers et des polynômes) est à la base de nombreuses applications en cryptographie. Nous commencerons par exposé les commandes usuelles de Maple avant de revenir sur les algorithmes fondamentaux de l'arithmétique dans $\mathbb{Z}$ (Euclide, Euclide étendu, etc.) et de les traduire en procédures. Une excellente référence pour cette seconde partie du TP est le **Cours d'Algèbre (primalité, divisibilité, codes) de Michel Demazure** *paru aux éditions Cassini.*

1. Principales commandes d'arithmétique

Maple peut manipuler des nombres comportant un très grand nombre de chiffres mais néanmoins limité. Le nombre maximal de chiffres, *maxdigits*, dépend du matétiel utilisé. On y aura accès par la commande **kernelopts(maxdigits)**.

```
> kernelopts(maxdigits);
```

$$268435448$$

Voici un aperçu des différentes opérations possibles sur les nombres entiers :

► *La fonction factorielle* : pour tout $n \in \mathbb{N}$, $n!$ se calcule au moyen de la commande **factorial** ou à l'aide du symbole **!**.

```
> factorial(12), 12!;
```

$$479001600, 479001600$$

► *Division euclidienne* : rappelons rapidement le principe de la division euclidienne : pour tous entiers naturels n et $m \neq 0$, il existe un unique couple d'entiers naturels (q,r) vérifiant $n = mq + r$ *et* $0 \leqslant |r| < |m|$. On dit que q et r sont respectivement le quotient et le reste dans la division euclidienne de n par m. On utilise les commandes **irem** et **iquo** pour calculer *le* reste et *le* quotient de la division euclidienne [1].

```
> irem(23,3);
                                    2
> irem(-3,2);
                                   -1
> -3 mod 2;
                                    1
> iquo(23,3);
                                    7
```

► *Test de primalité* : la fonction **isprime** permet de tester si un entier est premier [2] ou non.

```
> isprime(23);
                                 true
```

► *Décomposition en produit de facteurs premiers* [3] : la fonction **ifactor** calcule la décomposition en produit de facteur(s) premier(s) d'un entier donné.

```
> ifactor(561);
                              (3)(11)(17)
```

On a donc $561 = 3 \times 11 \times 17$.

1. **irem** et **iquo** sont les abréviations respectives de *integer remainder* et *integer quotient*.
2. Un nombre entier est dit premier s'il est ditinct de 1 et si ses seuls diviseurs sont 1 et lui-même.
3. Il s'agit du théorème fondamental de l'Arithmétique : tout nombre entier distinct de 1 se décompose en produit de facteurs premiers.

On rappelle rapidement les principales fonctions arithmétiques offertes par Maple.

Arithmétique élémentaire sous Maple

- **irem(n,m)** : renvoie un reste de n dans la division euclidienne par m.
- **iquo(n,m)** : renvoie un quotient de n dans la division euclidienne par m.
- **igcd(n,m)** : renvoie $\text{pgcd}(n,m)$.
- **igcdex(a,b,'u','v')** : affecte aux variables u et v deux entiers relatifs vérifiant $a \wedge b = u \cdot a + v \cdot b$.
- **ilcm(n,m)** : renvoie $\text{ppcm}(n,m)$.
- **ithprime(n)** : renvoie le n-ième nombre premier.
- **isprime(n)** : renvoie *true* si n est premier, *false* sinon.
- **ifactor(n)** : renvoie la décomposition en produit de facteur(s) premier(s) de n.

Exercice 1.

On note

$$S_3(\mathbb{Z}) = \left\{ x \in \mathbb{Z} \;\middle|\; \exists (x_1,x_2,x_3) \in \mathbb{Z}^3 \,,\; x = \sum_{k=1}^{3} x_k^2 \right\}$$

1. Ecrire un programme renvoyant la liste des 15 plus petits éléments (au sens de la relation d'ordre $\leqslant$) de $S_3(\mathbb{Z})$. En déduire que $S_3(\mathbb{Z})$ n'est pas stable par le produit.

2. Soit a, b, c et d quatre entiers relatifs tels que

$$a^2 + b^2 + c^2 + d^2 \equiv 0\,[8]$$

Vérifier qu'ils sont tous pairs.

3. Soit n un entier tel que $n \equiv -1\,[8]$. Montrer que $n \notin S_3(\mathbb{Z})$.

2. Les algorithmes fondamentaux de l'arithmétique

Cette partie expose les algorithmes élémentaires de l'arithmétique. On n'utilisera pas les commandes de Maple telles que `igcd`, `irem`, `iquo`, etc. Le principe est de redéfinir ces procédures au moyen des algorithmes.

2.1. La division euclidienne

Rappelons le théorème de la division euclidienne :

La division euclidienne dans $\mathbb{N}$

Soient a et b deux entiers naturels tels que $0 < b$. Il existe un unique couple d'entiers naturels (q, r) tel que

$$a = b \cdot q + r \quad \text{et} \quad 0 \leqslant r < b$$

appelés respectivement *quotient* et *reste* dans la division euclidienne de a par b. On dit aussi que r et q sont respectivement *le reste et le quotient de a modulo b*.

L'algorithme de la division euclidienne de a par b le plus élémentaire est fondé sur des soustractions successives de b au nombre a.

Algorithme de la division euclidienne

Initialisation $\begin{cases} r \leftarrow a \\ q \leftarrow 0 \end{cases}$

Tant que $r \geqslant b$ **faire** $\begin{cases} r \leftarrow r - b \\ q \leftarrow q + 1 \end{cases}$

Renvoyer q et r.

A la sortie de la boucle **Tant que**, les entiers q et r sont respectivement égaux au quotient et au reste dans la division euclidienne de a par b.

Exercice 2. *Division euclidienne*

Ecrire une procédure `DivEuclide(a,b)` d'arguments $a \in \mathbb{N}$ et $b \in \mathbb{N}^*$ renvoyant sous forme d'une liste $[q, r]$ le quotient et le reste dans la division euclidienne de a par b.

2.2. Les algorithmes d'Euclide

L'algorithme d'Euclide de calcul du pgcd de deux entiers naturels a et b est fondé sur la division euclidienne. On notera $a \wedge b$ le pgcd des entiers a et b. En voici le principe :

On note $r_0 = a$ et $r_1 = b$. Si $b = 0$, $r_0 \wedge r_1 = a$. Sinons, on effectue en cascade des divisions euclidiennes :

$$\left\{\begin{array}{lcccl} r_0 & = & r_1\cdot q_2+r_2 & \text{où} & r_2<r_1\\ r_1 & = & r_2\cdot q_3+r_3 & \text{où} & r_3<r_2<r_1\\ \vdots & \vdots & \vdots & \vdots & \vdots\\ r_i & = & r_{i+1}\cdot q_{i+2}+r_{i+2} & \text{où} & 0<r_{i+2}<r_{i+1}<r_i<\cdots<r_1\\ \vdots & \vdots & \vdots & \vdots & \vdots\\ r_{n_0-2} & = & r_{n_0-1}\cdot q_{n_0}+r_{n_0} & \text{où} & 0=r_{n_0}<r_{n_0-1}<\cdots<r_2<r_1 \end{array}\right.$$

Le processus s'arrête car les restes r_i forment une séquence strictement décroissante d'entiers naturels : il existe $n_0\in\mathbb{N}$ tel que $r_{n_0}=0$.

Venons-en à la remarque fondamentale : pour tout $0\leqslant i\leqslant n_0-2$, l'ensemble des diviseurs de r_i et r_{i+1} étant égal à l'ensemble des diviseurs de r_{i+1} et r_{i+2}, on a

$$a\wedge b=r_0\wedge r_1=r_1\wedge r_2=\cdots=r_{n_0-1}\wedge r_{n_0}=r_{n_0-1}$$

On retiendra que le pgcd de a et b est égal au dernier reste non nul dans l'algorithme d'Euclide.

Algorithme d'Euclide

Initialisation : $\left\{\begin{array}{l} r_0=a\\ r_1=b\end{array}\right.$

Tant que $r_1\neq 0$ **faire** $\left\{\begin{array}{l} x\leftarrow r_0\\ r_0\leftarrow r_1\\ r_1\leftarrow \text{reste dans la division euclidienne de } x \text{ par } r_0\end{array}\right.$

Renvoyer r_0

Exercice 3. *Algorithme d'Euclide*

Ecrire une procédure `Euclide(a,b)` d'arguments a et b dans $\mathbb{N}$ et renvoyant $a\wedge b$ par l'algorithme d'Euclide.

Une variante de cet algorithme permet, au-delà du calcul du pgcd d de a et b, d'obtenir les coefficients (u,v) dans $\mathbb{Z}$ d'une relation de Bezout :

$$u\cdot a+v\cdot b=d$$

En voici le principe en reprenant les notations précédentes. On prouve par une récurrence facile que, pour tout $0\leqslant n\leqslant n_0$, il existe $(u_n,v_n)\in\mathbb{Z}^2$ tel que

$$r_n=u_n\cdot a+v_n\cdot b$$

En effet si cette propriété est vérifiée pour tout $k\leqslant n$, on a

$$\left\{\begin{array}{l} r_n=u_n\cdot a+v_n\cdot b\\ r_{n+1}=u_{n+1}\cdot a+v_{n+1}\cdot b\end{array}\right. \quad\text{et}\quad r_{n+2}=r_n-q_{n+2}\cdot r_{n+1}$$

ainsi

$$\begin{aligned} r_{n+2} &= r_n - q_{n+2}\cdot r_{n+1} = u_n\cdot a + v_n\cdot b - q_{n+2}\cdot(u_{n+1}\cdot a + v_{n+1}\cdot b) \\ &= \big(u_n - q_{n+2}\cdot u_{n+1}\big)\cdot a + \big(v_n - q_{n+2}\cdot v_{n+1}\big)\cdot b \end{aligned}$$

On peut donc définir une telle séquence de couple par les relations de récurrence suivantes :

$$\begin{cases} u_0 = 1 \\ v_0 = 0 \\ u_1 = 0 \\ v_1 = 1 \end{cases}, \quad \forall 0 \leqslant n \leqslant n_0 - 2, \quad \begin{cases} u_{n+2} = u_n - q_{n+2}\cdot u_{n+1} \\ v_{n+2} = v_n - q_{n+2}\cdot v_{n+1} \end{cases}$$

Le couple recherché est (u_{n_0-1}, v_{n_0-1}) puisque

$$a \wedge b = r_{n_0-1} = u_{n_0-1}\cdot a + v_{n_0-1}\cdot b$$

Cet algorithme est appelé *algorithme d'Euclide étendu* :

Algorithme d'Euclide étendu

Initialisation : $\begin{cases} r_0 \leftarrow a \\ r_1 \leftarrow b \end{cases}$, $\begin{cases} u_0 \leftarrow 1 \\ u_1 \leftarrow 1 \end{cases}$, $\begin{cases} v_0 \leftarrow 0 \\ v_1 \leftarrow 1 \end{cases}$

Tant que $r_1 \neq 0$ **faire** $\begin{cases} x \leftarrow r_0 \\ r_0 \leftarrow r_1 \\ r_1 \leftarrow \text{reste de } x \text{ modulo } r_0 \\ y \leftarrow u_0 \\ u_0 \leftarrow u_1 \\ u_1 \leftarrow y - u_0 \times \text{quotient de } x \text{ modulo } r_0 \\ y \leftarrow v_0 \\ v_0 \leftarrow v_1 \\ v_1 \leftarrow y - u_0 \times \text{reste de } x \text{ modulo } r_0 \end{cases}$

Renvoyer (u_0, v_0)

Exercice 4. *Algorithme d'Euclide étendu*

Ecrire une procédure `EuclideEtendu(a,b)` d'arguments a et b dans $\mathbb{N}$ et renvoyant un couple $(u, v) \in \mathbb{Z}^2$ tel que $a \wedge b = u\cdot a + v\cdot b$.

2.3. L'écriture en base b des entiers naturels

Soit $b \in \mathbb{N} \setminus \{0, 1\}$. Tout entier naturel n peut s'écrire de manière unique sous la forme

$$n = c_m\cdot b^m + c_{m-1}\cdot b^{m-1} + \cdots + c_1\cdot b + c_0$$

où les c_k valent $\{0, 1, \cdots, b-1\}$ avec $c_m \neq 0$. Cette expression est appelée *écriture de n en base b* ; les c_k sont les chiffres de n en base b. Dans le cas particulier où $b = 10$, on retrouve l'écriture décimale usuelle des entiers : c_0 est le chiffre des unités de n, c_1 celui des dizaines, etc. Dans le cas d'une base b quelconque, on notera de manière condensée $n = (c_m\, c_{m-1} \cdots c_1\, c_0)_b$.

Rappelons la preuve de ce résultat de numération. Elle est intéressante car constructive, c'est-à-dire qu'on en déduit un algorithme explicite de calcul des chiffres de n en base b. L'idée repose sur l'égalité suivante :

$$n = \Big(\Big(\Big(\cdots\big((c_m)\times b + c_{m-1}\big)\cdots\Big)\times b + c_2\Big)\times b + c_1\Big)\times b + c_0$$

Le couple quotient-reste dans la division euclidienne de n par b est de la forme (q_0, c_0), le couple quotient-reste dans la division euclidienne de q_0 par b est de la forme (q_1, c_1) et ainsi de suite. Ceci nous éclaire et nous allons considérer la suite de couples (q_p, c_p) définie par :

$$\begin{cases} (q_0, c_0) &= \big(\text{quotient de } n \text{ modulo } b, \text{reste de } n \text{ modulo } b\big) \\ (q_{i+1}, c_{i+1}) &= \big(\text{quotient de } q_i \text{ modulo } b, \text{reste de } q_i \text{ modulo } b\big) \quad \text{pour tout } i \in \mathbb{N} \end{cases}$$

Comme $0 \leqslant q_{i+1} < q_i$ ou $q_i = q_{i+1} = 0$ pour tout entier naturel[4] i, la suite $(q_i)_{i\geqslant 0}$ est nulle à partir d'un certain rang i_0. En notant m le plus petit entier m vérifiant cette propriété, on a donc que $c_i = 0$ pour tout $i \geqslant m+1$ et $c_m = q_m \cdot b + c_m = q_{m-1} =\neq 0$. On prouve alors par une récurrence facile que

$$\forall i \in \mathbb{N},\ \ n = q_i \cdot b^{i+1} + \sum_{k=0}^{i} c_k \cdot b^k$$

d'où, en particulier, pour $i = m$,

$$n = \sum_{k=0}^{m} c_k \cdot b^k$$

Le cas de la base $b = 2$ est particulièrement car c'est sous cette forme que la plupart des ordinateurs manipulent les nombres entiers.

Algorithme de calcul de l'écriture en base b d'un entier naturel

Intialisation : $\begin{cases} q_0 \leftarrow \text{quotient de } n \text{ modulo } b \\ c_0 \leftarrow \text{reste de } n \text{ modulo } b \\ L \leftarrow \text{liste à un seul élément } c_0 \end{cases}$

Tant que $q_0 \neq 0$ **faire** $\begin{cases} x \leftarrow q_0 \\ q_0 \leftarrow \text{quotient de } x \text{ modulo } b \\ c_0 \leftarrow \text{reste de } x \text{ modulo } b \\ \text{insérer } c_0 \text{ au début de la liste L} \end{cases}$

Renvoyer L

Exercice 5.

Ecrire une procédure `EcritureBinaire(n)` prenant en argument l'entier naturel n et renvoyant la liste $[c_m, \ldots, c_1, c_0]$ de ses chiffres en base 2.

4. En effet, $q_i = q_{i+1}\cdot b + c_{i+1}$ avec b et c_{i+1} positifs implique $q_{i+1} \leqslant q_i$. D'autre part, l'égalité $q_{i+1} = q_i$ équivaut à $(1-b)\cdot q_i = c_{i+1}$ et puisque les deux membres sont de signes opposés, ils sont nuls ; on en déduit que $q_i = 0$ car $b \neq 1$.

2.4. Le calcul rapide des puissances

Commençons par un exemple élémentaire, le calcul de 4^6. On peut commencer par une méthode naïve d'élévation à la puissance 6 :

$$\begin{aligned}
4^2 &= 4\times 4 = 16\\
4^3 &= 4^2\times 4 = 16\times 4 = 64\\
4^4 &= 4^3\times 4 = 64\times 4 = 256\\
4^5 &= 4^4\times 4 = 256\times 4 = 1024\\
4^6 &= 4^5\times 4 = 1024\times 4 = 4096
\end{aligned}$$

On a effectué au total 5 multiplications. On peut améliorer ce calcul de la manière suivante :

$$\begin{aligned}
4^2 &= 4\times 4 = 16\\
4^4 &= 4^2\times 4^2 = 16\times 16 = 256\\
4^6 &= 4^4\times 4^2 = 1024\times 4 = 4096
\end{aligned}$$

Cette séquence est plus efficace car ne nécessite que 3 multiplications. En réfléchissant un peu, on s'aperçoit qu'elle repose sur l'écriture en base 2 de 6. On a $6 = (110)_2$ et

$$4^6 = 4^{2^2+2} = \left(4^1\right)^{2^2}\times\left(4^1\right)^{2^1}\times\left(4^0\right)^{2^0}$$

De même, pour $9 = (1001)_2$, on a effectivement

$$4^9 = \left(4^1\right)^{2^3}\times\left(4^0\right)^{2^2}\times\left(4^0\right)^{2^1}\times\left(4^1\right)^{2^0}$$

Plus généralement, si $n = (n_p\cdots n_0)_2$, on a clairement

$$4^n = \left(4^{n_p}\right)^{2^p}\times\left(4^{n_{p-1}}\right)^{2^{p-1}}\times\cdots\times\left(4^{n_0}\right)^{2^0} = \left(\left(\left(\cdots\left(\left(1\times 4^{n_p}\right)^2\times 4^{n_{p-1}}\right)^2\cdots\right)^2\times 4^{n_2}\right)^2\times 4^{n_1}\right)^2\times 4^{n_0}$$

Au total, on dénombre $2p$ multiplications avec cette méthode. On en déduit un calcul de proche en proche de 4^n selon le principe suivant :

$$\begin{cases} z_0 &= 1\\ z_{i+1} &= z_i^2\cdot 4^{n_{p-i}} \quad \text{pour tout } 0\leqslant i\leqslant p\end{cases}$$

Exercice 6.

Ecrire une procédure `PuissancesRapides1(a,n)` calculant a^n par cette méthode (avec $n\in\mathbb{N}$ et $a\in\mathbb{C}$). On utilisera la procédure `EcritureBinaire(n)` précédemment programmée.

Cette approche peut cependant être améliorée : la connaissance de l'écriture binaire de l'entier naturel n n'est pas indispensable. On remarque simplement que

$$a^n = \begin{cases} a & \text{si } n = 1 \\ \left(a^{n/2}\right)^2 & \text{si } n \text{ est pair} \\ a \times \left(a^{(n-1)/2}\right)^2 & \text{si } n \geqslant 3 \text{ et } n \text{ impair} \end{cases}$$

Algorithme récursif de calcul rapide des puissances

$$Puissance(a,n) \leftarrow \begin{cases} a & \text{si } n = 1 \\ \left(Puissance(a,n/2)\right)^2 & \text{si } n \text{ est pair} \\ a \times \left(Puissance(a,(n-1)/2)\right)^2 & \text{si } n \text{ est impair et } n \geqslant 3 \end{cases}$$

Exercice 7. *Algorithme de calcul rapide des puissances*

Ecrire une procédure `PuissancesRapides2(a,n)` calculant a^n où $n \in \mathbb{N}$ et $a \in \mathbb{C}$ par l'algorithme de calcul rapide des puissances exposé ci-dessus.