Algèbre et arithmétique - mat309

B. Parisse

Septembre 2021

URL de ce diaporama

`https://www-fourier.univ-grenoble-alpes.fr/~parisse/mat249/m309cours1.pdf`

Contenu

Ce cours présente des mathématiques avec une forte composante algorithmique, qui sont utilisées dans des domaines comme la cryptographie ou la détection et correction d'erreurs de transmission.

1. Motivations : cryptographie, RSA, codes de Hamming binaires (hors programme cette année)
2. Division euclidienne et écriture en base, opérations + - * sur de grands entiers, liens avec les polynômes
3. Divisibilité, PGCD, algorithme d'Euclide et Euclide étendu, nombres premiers, restes chinois
4. L'anneau $\mathbb{Z}/n\mathbb{Z}$ et le corps $\mathbb{Z}/p\mathbb{Z}$ (p premier)
5. Calcul matriciel sur $\mathbb{Z}/p\mathbb{Z}$ ($p = 2$) et applications

Organisation

- 21 h de cours, 14 séances de 1h30. Il y a 2 créneaux de cours par semaine, cela permettra de dédoubler le cours si nécessaire.
- 36h de TD, 24 séances de 1h30.
- Site chamilo `https://chamilo.univ-grenoble-alpes.fr/courses/GBX3MT39/index.php`

Évaluation

1. CC1 : partiel semaine du 8 novembre
2. CC2 : interrogations en TD
3. ET1 : janvier 2022
4. ET2 : 2ème chance juin 2022
5. Règle du max si ET est supérieur à la moyenne des CC

Un formulaire manuscrit recto-verso est autorisé au CC1 et aux ET. Les calculatrices sont interdites aux interrogations, elles sont autorisées au CC1 et aux ET. Il est conseillé d'avoir une calculatrice graphique, si possible avec calcul exact sur des entiers sans limitation de taille.

Ressources

1. Page web des ressources de l'UE et cours
   `https://www-fourier.univ-grenoble-alpes.fr/~parisse/#mat309`
   `https://www-fourier.univ-grenoble-alpes.fr/~parisse/mat249/mat309.html`
2. sous-forum UGA-LST du Forum de Xcas
   `https://xcas.univ-grenoble-alpes.fr/forum/`
   login générique `etuinf`, password indiqué sur Chamilo
3. outil de calcul en ligne (Xcas ou MicroPython)
   `https://www-fourier.univ-grenoble-alpes.fr/~parisse/xcasfr.html`
4. Si cours à distance nécessaire, site chamilo ou : `https://meet.univ-grenoble-alpes.fr/b/ber-uph-xdu`

Cryptographie monoalphabetique à clef secrète.

- Décalage circulaire de l'alphabet (César), exemple.
- Variante avec n'importe quelle permutation des lettres de l'alphabet.
- Cryptographie symétrique : la connaissance de la clef de cryptage donne la clef de décryptage.
- Attaques ? Force brute, fréquences des lettres selon la langue.
- Méthodes plus robustes : on n'utilise pas toujours la meme lettre pour une lettre donnée, mais cela nécessite l'échange préalable de la clef. Cf. *The Key to Rebecca (Le Code Rebecca)* de Ken Follett basée sur l'histoire de l'espion nazi Johannes Eppler et l'opération Salaam. L'attaque d'Enigma par Alan Turing, *The Imitation Game*.

Cryptographie à clef publique.

Ce sont des systèmes asymétriques ne nécessitant pas d'échange préalable de clefs, RSA est l'exemple le plus connu.

- Numérisation de l'information à transmettre, un entier a.
- Choix des clefs $N = pq$ avec p, q premiers, et c et d tels que le reste de cd par $(p-1)(q-1)$ soit 1. N et c sont publics, p, q, d sont secrets.
- codage $b = a^c \pmod N$ est le message crypté, décodage on montre (maths !) que $a = b^d \pmod N$
- La sécurite repose sur la difficulté de factoriser des entiers vs les multiplier.
- L'algorithme de codage/décodage doit être rapide,
- Il est essentiel d'évaluer l'efficacité des algorithmes.

Illustration

- décrypter le code de César
  `IZIRSZVEHLZIZIRCVUVIEZVI`
- RSA jouet $p = 17, q = 19, N = 323, c = 11, d = 131$. Prendre un entier $a < 323$, calculer $b = a^c \pmod N$ puis $b^d \pmod N$, on doit retrouver a. Attention au dépassement de capacité selon le logiciel utilisé ! En Xcas

```
p:=17; q:=19; N:=p*q;
c:=11; d:=131; irem(c*d,(p-1)*(q-1))
a:=randint(N);
b:=pow(a,c,N); pow(b,d,N)
```

  En Python, remplacer `irem` préfixé par `%` infixé.

Détection et correction d'erreur

La transmission d'information sur un canal bruité génère des erreurs, qu'il faut pouvoir détecter et si possible corriger.

- Bit de parité : 7 bits d'information, on ajoute 1 bit pour avoir un nombre pair de bits à 1, détection mais pas de correction
- Code de répétition : on répète 3 fois chaque bit. On peut alors corriger à la majorité. Mais cela coute 3 fois plus de bande passante.
- Calcul matriciel modulo 2 permet de faire mieux : codes de Hamming binaire.

Codes de Hamming binaire

- on choisit $m > 1$, on pose $n = 2^m - 1$ et $k = n - m$. Par exemple $m = 3, n = 7, k = 4$.
- L'information est codée par un vecteur v ayant k bits
- On transmet w un vecteur de n bits : d'abord les k bits de v puis m bits calculé en faisant le produit $C * v$, où C est une matrice fixée ayant k colonnes et $m = n - k$ lignes
- La personne qui recoit w teste si $C * w[0 : k] + w[k : n]$ est nul. Si ce n'est pas le cas il y a erreur.
- Pour pouvoir corriger une erreur, on choisit C tel que les colonnes de la matrice $(C|I_m)$ soient toutes non nulles et distinctes, en écrivant en base 2 les entiers de 1 à $n - 1$ et en permutant les colonnes de I_m à la fin.

Illustration

- Écrire les entiers de 1 à 7 en base 2 en colonne
- Permuter les colonnes de la matrice identité I_3 à la fin
- Extraire la première partie C, puis coder par exemple $v = [1, 0, 1, 1]$ en $w = (v, C * v)$ modulo 2.
- Vérifier que $H * w = 0$ (modulo 2) où $H = (C|I_3)$.
- Bonus : Simuler une erreur et sa correction. (On peut remarquer que $H * w$ est l'écriture en base 2 du numéro de bit à modifier après permutation).

Une solution avec Xcas

```
m:=[convert(j+9,base,2) for j in range(7)];
M:=tran(m)[0..2];
C:=tran(select(x->sum(x)!=1,tran(M)));
v:=[1,0,1,1] mod 2;
w:=concat(v,C*v);
H:=concat(C,identity(3));
H*w;
w[3]:=1-w[3]; H*w;
```

La première ligne utilise une astuce pour forcer la présence de 0 non significatifs de l'écriture en base 2 pour $n < 4$: on ajoute 8 qui crée un bit 1 en plus, qu'on enlève à la ligne suivante. L'extraction de C se fait en regardant si la somme des coefficients d'une colonne (ligne de la transposée) vaut 1.

Représentation des entiers

- Les régistres d'un microprocesseur peuvent stocker et manipuler des entiers de taille limitée. Les opérations arithmétiques sont faites modulo une puissance de 2.
- Par exemple entier signé dans $[-2^{31}, 2^{31} - 1]$ sur un processeur 32 bits, entier non signé dans $[0, 2^{64} - 1]$ sur un 64 bits.
- Pour les entiers plus grands, il est nécessaire d'utiliser une zone mémoire de plusieurs mots, il s'agit d'un cas particulier d'écriture d'un entier en base b. Le fondement mathématique est le
- Théorème : Soient a et b sont deux entiers naturels, $b \neq 0$. Il existe un unique couple q, r tel que $a = bq + r$, $r \in [0, b[$, ce sont le quotient et le reste euclidien de a par b.

Écriture en base b

- Soit $b \geq 2$ fixé. Tout entier $a > 0$ s'écrit de manière unique

$$a = \sum_{i=0}^{n} a_i b^i, \quad a_i \in [0, b[, a_n \neq 0$$

- Algorithme de calcul des a_i en fonction de a.
  Faire la division euclidienne de a par b, si le quotient q est nul alors $n = 0$ et $a_0 = a$. Sinon déterminer la décomposition en base b de q, décaler les indices de 1 et poser $a_0 = r$ le reste de la division.
- Algorithme de calcul de a connaissant les a_i.
  C'est un cas particulier de l'algorithme d'évaluation du polynôme $P(x) = \sum_{i=0}^{n} a_i x^i$ en $x = b$, qui se fait de manière optimale par la méthode de Horner.

Exemple

- Écrire 2020 en base 2, 3, 16.
- Faire l'opération réciproque.
- Vérification avec Xcas

```
l:=convert(2020,base,3)
convert(l,base,3)
```

- Dans le monde, il y a 10 catégories de personnes : celles qui connaissent le binaire et celles qui ne le connaissent pas.

Algorithme de Horner

- $P(x) = \sum_{i=0}^{n} a_i x^i$ est donné par la liste `l` de ses coefficients $[a_n, ..., a_0]$. On veut calculer $P(b)$. On écrit :

$$P(b) = ((...((a_n b + a_{n-1})b + a_{n-2})b + ...)b + a_0$$

- Algorithme de Horner en langage naturel
  Arguments : la liste `l` et le réel `b`
  `n` $\leftarrow$ longueur(`l`)-1
  `r` $\leftarrow$ `l[0]`
  Pour `j` de `1` jusque `n` faire
    `r` $\leftarrow$ `r*b+l[j]`
  Finpour
  Renvoyer `r`