Université de Bordeaux
Mathématiques

Algèbre et calcul formel – Agrégation
2022–2023

FEUILLE D'EXERCICES n° 7

Algorithme d'Euclide et applications

Exercice 1 – [Algorithme d'Euclide]
Programmer l'algorithme d'Euclide `AE` et l'algorithme d'Euclide étendu `AEE`. À l'aide de votre fonction AEE, programmer la fonction `Inverse(k,n)` qui pour deux entiers k et n premiers entre eux calcule l'inverse de k modulo n. Quelle est la complexité (en nombre d'opérations sur les bits) de votre fonction `Inverse` ? (On aura le droit de supposer que la fonction `AEE` a la même complexité que celle vue dans le cours.)

Bien sûr, ce n'est qu'un exercice : sur sage, on peut par exemple utiliser la commande `mod(k,n)**(-1)`, ou bien `mod(k,n)**(-1).lift` si l'on veut le résultat dans $\mathbb{Z}$.

Exercice 2 – [Autour de Bézout]

1) Le logiciel sage permet de calculer le pgcd de deux entiers à l'aide de la commande `gcd`. La commande `xgcd` donne en plus les coefficients de Bézout. En utilisant cette commande, écrire un algorithme qui prend en entrée une liste d'entiers $[a_1, \dots, a_n]$ et donne en sortie $[u_1, \dots, u_n] \in \mathbb{Z}^n$ tel que

$$\sum_{i=1}^{n} a_i u_i = \mathrm{pgcd}(a_1, \dots, a_n).$$

Quelle est la complexité de cet algorithme (en nombre d'opérations binaires, en supposant que tous les a_i sont en valeur absolue $\leqslant N$ pour un un certain entier $N > 0$ dont on se servira pour exprimer la complexité) ?

2) À partir du résultat donné par l'algorithme d'Euclide étendu, trouver une matrice U entière de déterminant ± 1 telle que $(a, b)U = (\mathrm{pgcd}(a, b), 0)$. Quelle est la complexité de cet algorithme (en nombre d'opérations binaires) ?

3) En s'inspirant de la question précédente, montrer qu'il existe une matrice U dans $\mathrm{GL}(n, \mathbb{Z})$ (entière, de déterminant ± 1) telle que

$$(a_1, \dots, a_n)U = (\mathrm{pgcd}(a_1, \dots, a_n), 0, \dots, 0),$$

et programmer le calcul de cette matrice.
(Indice : ne traiter que 2 coordonnées à la fois)
Quelle est la complexité de cet algorithme (en nombre d'opérations binaires) ?

Note. Cette question est une version "concrète" (d'un cas particulier) du théorème des diviseurs élémentaires pour les modules de type fini sur les anneaux principaux [*appliqué au dual* $(\mathbb{Z}^n)^*$ *et au sous* $\mathbb{Z}$*-module engendré par la forme linéaire* $(a_1, \dots, a_n)$]. Les diviseurs élémentaires sont donnés par le membre de droite.

4) Si b et $(a_1, \ldots, a_n)$ sont des entiers fixés, expliquer comment résoudre l'équation $\sum a_i x_i = b$ en nombre entiers (x_i) en utilisant l'algorithme de la question précédente (on veut trouver toutes les solutions). Résoudre les équations $1009x+345y+56z = 1$ et $143x+195y+165z = 3$.

Exercice 3 – [Restes Chinois]

1) Résoudre les systèmes

$$\begin{cases} x \equiv -1 \mod 21 \\ x \equiv 11 \mod 15 \\ x \equiv 1 \mod 10 \end{cases} \text{ et } \begin{cases} x \equiv -1 \mod 21 \\ x \equiv 10 \mod 15 \\ x \equiv 1 \mod 10 \end{cases}$$

(on pourra utiliser la commande `crt`).

Note. Cette commande `crt` s'applique à tout anneau où l'algorithme des restes chinois s'applique, par exemple à $k[x]$, où k est un corps.

On connaît bien l'algorithme correspondant dans le cas où les modules sont deux à deux premiers entre eux. Il ne s'applique pas tel quel aux exemples précédents. Les questions suivantes portent sur le cas général.

2) Soient a et b deux entiers > 0. On considère le système d'inconnue N

$$\begin{cases} N \equiv \alpha \pmod{a} \\ N \equiv \beta \pmod{b} \end{cases}$$

et on pose $\delta = \mathrm{pgcd}(a, b)$, puis u et v deux entiers tels que $au + bv = \delta$.

a) Montrer que le système n'a pas de solution si $\alpha \not\equiv \beta \pmod{\delta}$.

b) Sinon, montrer que

$$N := \alpha + u\frac{a}{\delta}(\beta - \alpha) = \beta + v\frac{b}{\delta}(\alpha - \beta) = u\frac{a}{\delta}\beta + v\frac{b}{\delta}\alpha$$

convient. Montrer que cette solution est unique modulo ab/δ.

Note. c'est une généralisation du "théorème chinois" au cas ou les modules ne sont pas nécessairement premiers entre eux. Si $\delta = \mathrm{pgcd}(a, b) = 1$, on trouve bien un isomorphisme (explicite) entre $\mathbb{Z}/a\mathbb{Z} \times \mathbb{Z}/b\mathbb{Z}$ et $\mathbb{Z}/ab\mathbb{Z}$ (qui à (α, β) associe N).

3) En déduire un algorithme pour résoudre un nombre quelconque de congruences simultanées, et le programmer. Il s'agit là d'un exercice. Pour ce calcul, il faudra ensuite utiliser la commande `crt` (comme dans la première question de l'exercice).
Quelle est la complexité binaire de l'algorithme ? (Comme d'habitude, on suppose que tous les entiers donnés en entrée sont $\leqslant N$.)

4) Après une série de rapines, une troupe de 14 pirates partage (équitablement) le butin et laisse le reliquat, 3 écus, au cuisinier chinois, le $15^{\text{ème}}$ homme d'équipage. Le lendemain, un flibustier tombe à la mer et n'est pas repéché à temps ; après avoir envisagé le versement de sa part à des œuvres, les pirates refont le partage en incluant sa part ; le cuistot reçoit 2 écus. La semaine se passe sans encombres, mais trois pirates ivres se disputent sur leurs parts respectives et deux d'entre eux sont tués. Notre cuisinier récupère 5 écus. La fin du

mois est mauvaise et 3 pirates périssent dans une embuscade ; mais le cuistot est content : il garde ses 5 écus.

Quel magot peut-il espérer empocher quand il décide d'empoisonner le reste de la bande ?

Exercice 4 – [Déterminant modulaire]
Soit $A \in M_n(\mathbb{Z})$. On se propose de calculer $\det A$ de façon modulaire, c'est-à-dire de calculer $\det A \pmod p$ à partir des $a_{i,j} \pmod p$, pour des p premiers et petits, et d'en déduire la valeur de $\det A$. Pour cela il faudra prendre des p dont le produit est plus grand que $2 \cdot |\det A|$ et se servir du lemme chinois. On rappelle l'inégalité de Hadamard :

$$|\det A|^2 \leqslant \prod_{j=1}^{n} \Big(\sum_{i=1}^{n} a_{i,j}^2 \Big),$$

qui aidera à déterminer une famille de p adaptée.

1) Écrire une procédure admettant en entrée A et qui détermine successivement une famille appropriée de premiers, les déterminants modulaires puis le déterminant de A. Faire attention aux fonctions `mod`$(*, d)$ ou $*\%d$ qui renvoient un entier de l'intervalle $[0, d\,[$ et non de $]-\frac{d}{2}, \frac{d}{2}]$. Quelle est la complexité binaire de votre algorithme (en fonction de $\max_{i,j} |a_{i,j}|$ et de n ?)

Remarque : L'intérêt de calculer le déterminant modulo des premiers, c'est qu'on n'a pas besoin de s'inquiéter de l'explosion de la taille des coefficients. Cela rend le calcul de la complexité binaire plus facile.

2) Tester sur $A \in M_{10}(\mathbb{Z})$ où les $a_{i,j}$ sont aléatoires et vérifient $|a_{i,j}| < 100$. Étendre les tests.

Exercice 5 – [Faire des expériences pour mesurer le temps]
On a vu en cours un algorithme pour calculer pgcd de deux entiers a et $b \leqslant N$ avec une complexité binaire $O(\log(N)^2)$. L'objectif de cet exercice est de déterminer si l'algorithme implémenté par Sage se comporte asymptotiquement en $\log(N)^2$, ou s'il fait mieux.

1) Écrire une fonction qui prend en entrée deux entiers N et k. L'algorithme répète k fois : tirer a et b au hasard inférieurs à N, puis mesurer le temps pour calculer leur pgcd. À la fin, l'algorithme renvoie le temps moyen pour le calcul d'un pgcd (en moyennant sur les k essais).

Pour mesurer le temps, on pourra utiliser la commande `import time` (à utiliser une seule fois), puis la commande `t = time.time()`, qui donne le temps à un moment de l'execution. En faisant

```
t1 = time.time()
bla
blo
t2 = time.time()
print(t2-t1)
```

on affiche le temps qu'il s'est écoulé entre le début et la fin des commandes `blablo`.

2) Se servir de la fonction précédente pour faire des expériences avec N de plus en plus grand et mesurer le temps T correspondant.

On veut ensuite tracer les résultats, quelle échelle vaut-il mieux utiliser pour vérifier que la complexité est quadratique ? (I.e., est-ce qu'il faut afficher T, $\log T$, $\log\log T$, ... en fonction de N, $\log(N)$, $\log\log N$,... ?)

Pour tracer des points, on peut définir un objet graphique `g = Graphics()`. Ensuite on peut ajouter des points à cet objet graphique en faisant `g = g+Point([x,y])` (on peut rajouter un argument `color = 'red'` ou autre). Et enfin, on peut afficher `g` en faisant `plot(g)`. On peut aussi tracer des lignes avec la commande `line`.

3) Conclure : l'algorithme de Sage est-il quadratique ?