Anneaux $\mathbb{Z}/n\mathbb{Z}$

Exercice 1 *Restes chinois*

1. Écrire une procédure permettant de résoudre le système de r congruences

$$x \equiv \nu_i \,(\mathrm{mod}\, m_i)\,,\ 1 \leqslant i \leqslant r\,.$$

où les m_i sont des entiers naturels 2 à 2 premiers entre eux.

(Voir la commande `crt` pour vérifier.)

2. Soient $(a_i)_{1\leqslant i\leqslant r}$ et $(\nu_i)_{1\leqslant i\leqslant r}$ deux suites de rationnels avec $a_i \neq a_j$ si $i \neq j$. En utilisant l'algorithme des restes chinois, écrire une procédure permettant de déterminer le polynôme interpolateur de Lagrange $P \in \mathbf{Q}[X]$ de degré au plus $r-1$ tel que

$$P(a_i) = \nu_i\,,\ \text{pour } 1 \leqslant i \leqslant r\,.$$

(En pratique on implantera d'abord l'algorithme des restes chinois « général » dans $\mathbf{Q}[X]$ pour répondre ensuite au cas particulier demandé.)

Exercice 2 *Cryptographie RSA*

On représente les 26 lettres de l'alphabet (en majuscule) en utilisant les entiers de 2 à 27 (on utilise 1 pour l'espace, et, par commodité, 0 correspond à un caractère non utilisé, par exemple %). On utilise les entiers de 28 à 54 pour les lettres de l'alphabet en minuscule. On dispose ainsi d'un « alphabet modifié » à 54 caractères.

On veut, en fait, étant donné un (grand) entier n, pouvoir faire correspondre de manière unique un élément de $\mathbf{Z}/n\mathbf{Z}$ à un mot donné. Cela n'est possible que si le mot en question a au plus k lettres : $w_0 w_1 \cdots w_{k-1}$ où k est le plus grand entier tel que $54^k \leqslant n$. L'entier correspondant à ce mot est alors $\sum_{i=0}^{k-1} w_i 54^i$. Si l'on veut envoyer des mots plus longs, on découpe le message en une suite de mots ayant tous au plus k lettres.

1. En utilisant ce principe, on écrit une fonction `encode` prenant en entrée un mot m (ou une phrase) et un entier k (que l'on pourra renseigner ou qui pourra prendre une valeur par défaut (par exemple $k = 5$)) et renvoyant une liste de nombres ayant tous au plus k chiffres en base 54. (Voir les commandes `find` et `ZZ(x,b)`.)

   Réciproquement, et selon le même principe, on écrit une procédure `decode` prenant pour entrées un entier N (en base 10) et renvoyant en sortie la signification de N (en toutes lettres). (Voir les commandes `flatten` et `join`.)

   Ces procédures sont disponibles à l'url :

   `http://www.math.u-psud.fr/~lelievre/t/DonneesRSA.txt`

   Vous pouvez les récupérer, les tester, et les utiliser tel quel dans la suite de l'exercice, ou bien implanter des fonctions `encode` et `decode` de votre cru.

2. Écrire une procédure `RSA(p,q)` prenant en entrée deux nombres premiers $p \neq q$ et renvoyant un couple clé publique/clé privée RSA : $(n = pq, e)$, (n, d).

3. Écrire une procédure `encrypte(m,cle_pub)` prenant en entrée une phrase en toutes lettres m et une clé publique RSA (n, e) et renvoyant le message chiffré associé à m par le cryptosystème RSA relatif à (n, e). On pourra chercher à optimiser la longueur de chaque bloc de manière à ce que la liste de nombres obtenus ait le moins d'éléments possibles. On pourra utiliser la syntaxe `n, e = cle_pub` pour récupérer les deux parties de la clé au début de la fonction.
4. Écrire une procédure `decrypte(L, cle_priv)` prenant en entrée une liste L de nombres et une clé privée RSA (n, d) et renvoyant en sortie le message clair (en toutes lettres) correspondant.
5. Écrire une procédure `casse(cle_pub)` pour casser une clef RSA, c'est à dire passer d'une clef publique (n, e) à la clef privée (n, d) correspondante (utiliser `factor` ou `euler_phi`).
6. Casser les clés publiques RSA (n, e) et déchiffrer les listes de mots chiffrés correspondant que l'on trouvera à l'url :

   `http://www.math.u-psud.fr/~lelievre/t/DonneesRSA.txt`

Exercice 3 *Factorisation d'entiers*

1. Implanter l'algorithme ρ de Pollard. À l'entrée (n, N), la procédure fera correspondre la sortie $(p, q), N'$, où n est l'entier à factoriser dont on sait qu'il est produit des 2 nombres premiers distincts p et q, l'entier N désigne le nombre maximal d'essais que l'on autorise `Sage` à effectuer et N' est le nombre d'essais effectivement utilisés par `Sage` pour factoriser n (on entend par "essai" un choix pour la donnée initiale x_0 dans l'algorithme de Pollard).

   On pourra utiliser la commande `ZZ.random_element(k)`.
2. Implanter, à partir de la question précédente, un algorithme permettant de factoriser un entier sans facteur carré donné en produit de nombres premiers.

   On pourra utiliser la commande `is_prime`.
3. Écrire une procédure prenant un entier naturel r pour entrée et renvoyant un nombre premier aléatoire de taille r.

   [Indication : on pourra utiliser `next_prime`]
4. Écrire une procédure qui à l'entrée r (un entier naturel) fait correspondre un produit $p \times q$ de 2 nombres premiers distincts aléatoires de taille r.
5. Utiliser la question précédente pour tester l'algorithme de Pollard sur des entiers produits de 2 nombres premiers distincts dont la taille est de plus en plus grande. Pour quelle valeur de $2r$ la méthode commence-t-elle à montrer ses limites ? Comparer avec la commande `factor` de `Sage`.

Exercice 4 *Correction d'erreurs et restes chinois*

1. Soient a, b, m, p des entiers positifs tels que $a > b$ et $a \geqslant 4mp > 0$. Dans les notations de l'algorithme d'Euclide étendu pour le couple (a, b), soit i l'indice tel que $2m \in [r_i, r_{i-1}[$. Montrer que pour tout couple d'entiers (x, y) tel que $|xa + yb| \leqslant m$ et $0 < |y| \leqslant p$, il existe un entier u tel que

$$xa + yb = ur_i\,, \qquad x = us_i\,, \qquad y = ut_i\,.$$

   [Indication : on pourra commencer par montrer que dans la suite $(t_i)_i$ les signes alternent et que $|s_j t_{j-1} - s_{j-1} t_j| = 1$ pour tout j, puis déduire $a \geqslant |t_j| r_{j-1}$ pour tout j.]

2. Soient $n_1, \dots n_k$ des entiers positifs deux à deux premiers entre eux et $N := \prod_i n_i$. Soit $(a_1, \dots, a_k)$ un k-uplet d'entiers tels que pour tout j on ait $0 \leqslant a_j \leqslant n_j - 1$. Soit S l'unique entier de $[0, N[$ satisfaisant aux congruences $S \equiv a_i (\mathrm{mod}\, n_i)$. On suppose que l'on utilise un canal brouillé pour communiquer « l'information » $(a_1, \dots, a_k)$. On note $(b_1, \dots, b_k)$ le message reçu et $\ell := \#\{j : a_j \neq b_j\}$ le « poids » de l'erreur commise lors de la transmission.

   Notons $P = n_{i_1} \cdots n_{i_\ell}$, où, pour tout j, $a_{i_j} \neq b_{i_j}$. On suppose qu'il existe un entier positif M vérifiant $N \geqslant 4P^2M$ et $0 \leqslant S \leqslant M$. En d'autres termes on suppose que l'on a un bon contrôle du nombre maximal d'erreurs pouvant être commises durant la transmission et que S n'est pas trop grand. Montrer que l'on peut récupérer le message initial S à partir de la seule connaissance de N, B, P et M, où $B \in \{0, \dots N-1\}$ est l'unique entier vérifiant $B \equiv b_i (\mathrm{mod}\, n_i)$ pour tout i.

3. Simuler sous `Sage` une erreur de transmission vérifiant les hypothèses de la question précédente (on pourra utiliser la commande `ZZ.random_element(m)` pour choisir un entier au hasard dans $\{0, \dots, m-1\}$). En pratique on écrira une procédure `erreur(L,a)` prenant en entrée une liste `L` d'entiers deux à deux premiers entre eux et un entier `a` bornant l'erreur de transmission. Cette procédure devra d'abord générer un triplet `(S,M,P)` vérifiant les conditions de la question précédente (en particulier `S` est l'entier de $\{0, \dots, N-1\}$ à transmettre). On renverra en sortie `(B,M,P)` où `B` est un entier « erroné » de $\{0, \dots, N-1\}$ obtenu à partir de S en ajoutant au plus a à chaque reste dans la division euclidienne de S par les composantes de `L`.

4. Écrire une procédure `recupere(L,B,M,P)` (où l'on conserve les notations de la question précédente) permettant la correction du message `B` (i.e. permettant de retrouver `S`).

5. Quel rôle est joué par l'entier a bornant la taille de l'erreur de transmission commise ?