

Résumé

Ce devoir est une synthèse de nos travaux depuis plusieurs mois sur la reconstitution du théorème des restes chinois. Grace au soutiens de notre professeur MR ZEMOR, nous avons appris a reconstruire le théorème des restes chinois en présence d'erreur ; Dans un premier temps, nous avons pris connaissance de ce théorème, puis dans un second temps, nous l'avons corrigé grace à plusieurs méthode, tel que du brut force ou une méthode de fractions continues.

Avant de commencer, il faut établir des bases. Dans un premier temps quand nous parlons de borne, nous parlons du fait qu'après cette borne la recherche d'erreur ne marche plus. dans un second temps, les notations. Dans le théorème des restes chinois, nous travaillons avec des listes de nombres que l'on note P la liste des nombres premiers, et N la liste des restes modulo les P_i .

Table des matières

1	théorème des restes	4
1.1	théorème des restes chinois et son histoire	4
1.2	notre algorithme	6
2	Brute force	7
2.1	Notre premier algo version naïve	7
2.2	brute force de hamming	9
3	Les fractions continues	10
3.1	Principe	10
3.2	Résumé d'expériences et test	11

Nous tenons tout d'abord à remercier Mr CASTAGNOS Guillaume d'avoir été le professeur principale de cette matière et, en ayant bien expliqué le fonctionnement pour partir sur de bonne base. Nous remercions par la suite Mm ZEMOR Gilles pour nous avoir suivit, aider et tutoré durant ce semestre.

nous remercions également nos familles et nos proches, pour leurs aide et leur soutiens.

1 théorème des restes

1.1 théorème des restes chinois et son histoire

Le théorème des restes chinois viens à la base d'un livre mathématique chinois de Qin JUSHIO publié en 1247. Cependant, on avait déjà découvert ce théorème au part avant dans un livre de Sun ZI au 3° siècle. Le théorème consiste en : On pose p_1, \dots, p_k des entiers premiers 2 à 2. Pour tout n_1, \dots, n_k , il existe un entier x tel que :

$$x \equiv n_1 \pmod{p_1}$$

$$x \equiv n_2 \pmod{p_2}$$

$$x \equiv n_k \pmod{p_k}$$

Nous allons dans un premier temps démontrer l'unicité de ce théorème. d'abord, voici l'existence d'une solution :

On a $\forall k \in [1; y]$, $x \equiv n_k \pmod{p_k}$

$\forall k \in [1; y]$, on note $P_k = \frac{P}{p_k}$ où $P = p_1 \times \dots \times p_y$

On voit assez simplement que P_k et p_k sont premier entre eux car tout les entiers p_i sont premier entre eux 2 à 2. Et donc P_k est inversible modulo p_k , car en faisant le théorème d'euclide, on va trouver $u, v \in \mathbb{Z}$ tel que $P_k \times u + p_k \times v = d$ avec d le PGCD. Or P_k et p_k sont premier entre eux donc $d = 1$ et on trouve facilement l'inverse de P_k modulo p_k .

On note alors u_k le nombre tel que $u_k \times P_k + v_k \times p_k = 1$, soit $u_k \times P_k \equiv 1 \pmod{p_k}$. Soit $x = \sum_{k=1}^y u_k \times P_k \times n_k$. On pose $i \in [1; y]$ et $k \neq i$ alors $P_i \equiv 0 \pmod{p_k}$ car $P_i = p_1 \times \dots \times p_k \times \dots \times p_y$.

On a donc $x \equiv u_i P_i a_i \pmod{p_i}$, mais on sait que $u_i P_i \equiv 1 \pmod{p_i}$ d'où $x \equiv a_i \pmod{p_i}$. Sachant que c'est vrai pour tout i , on a x solution du système.

On a x une solution du système, posons y une autre solution du système. On a alors $x - y \equiv 0 \pmod{p_k}$, soit p_k divise $x - y$. Sachant que les p_k sont premier 2 à 2 entre eux, on a donc $x - y \equiv 0 \pmod{P}$, soit P divise $x - y$, ou encore que $x \equiv y \pmod{P}$, d'où l'unicité modulo P .

Pour illustrer ce théorème, nous allons donner un exemple, mais pas n'importe quelle exemple, celui dont Sun ZI a proposé une solution :

soient des objets, prenons des bonbons, si on les repartis pour 3 enfants, il en reste 2, si on les répartit pour les 3 enfants et leurs parents (soit 5 personnes), il en reste 3. enfin si on partage ces bonbons avec également les 2 cousins, (soit 7 personnes) il en reste 2. On a donc :

$$x \equiv 2 \pmod{3}$$

$$x \equiv 3 \pmod{5}$$

$$x \equiv 2 \pmod{7}$$

La question que l'on se pose à présent est combien y a t'il de bonbons ?

Grace au théorème des restes chinois, on peut trouver la réponse. Enfin, nous avons plusieurs réponse, c'est à dire que tout les nombres congru a x modulo n sont des bonnes réponses. Avant de répondre à ce problème, il faut d'abord voir l'algorithme.

1.2 notre algorithme

L'algorithme du théorème des restes chinois étant un peu lourd, nous allons montrer l'algorithme comme suivant, mais vous pouvez le voir plus en détailles en python dans l'annexe. On a ici :

Soit n_i le i -ème termes de la liste des modulus, et on note

$$N_i = \frac{n}{n_i} = n_1 n_2 \dots n_i - 1 n_i + 1 \dots n_k$$

on a donc N_i et n_i qui sont premier entre eux.

Il faut alors faire l'algorithme d'euclide étendu sur N_i et n_i , se qui nous donne :

$$1 = N_i u_i + n_i v_i \text{ où } u_i, v_i \in \mathbb{Z}$$

On pose donc $e_i = u_i N_i$ avec $e_i \equiv 1 \pmod{n_i}$ et $e_i \equiv 0 \pmod{n_j}$ avec $i \neq j$

On trouve donc une solution qui est $x = \sum_{i=1}^k p_i e_i$.

Nous pouvons à présent répondre à l'exemple précédent. Nous avons :

$$x \equiv 2 \pmod{3}$$

$$x \equiv 3 \pmod{5}$$

$$x_k \equiv 2 \pmod{7}$$

On a $N_1 = 5 \times 7 = 35$, $N_2 = 3 \times 7 = 21$, et $N_3 = 3 \times 5 = 15$

On fait l'algorithme d'euclide sur N_1 et n_1 qui donne $-3 \times 23 + 2 \times 35 \times 1 = 1$ et aussi $e_1 = 2 \times 35$

Idem sur N_2 et n_2 qui donne $21 \times 1 - 5 \times 4 = 1$, et donc $e_2 = 21$

Enfin, on a $15 \times 1 - 7 \times 2 = 1$ avec $e_3 = 15$

Le résultat final est donc $x = 2 \times 35 + 3 \times 21 + 2 \times 15 = 233$. Nous avons, comme dit précédement plusieurs résultat, qui sont les entiers congrus à 233 modulo 105

Comme vous pouvez le voir, nous avons utilisé l'algorithme d'Euclide étendu, qui prend en paramètres a et b, deux entiers, et qui renvoie d le PGCD de a et b, un couple $(u, v) \in \mathbb{Z}$ tel que $d = au + bv$.

Pour notre utilisation, nous avons un peu modifié cette algorithm, car pour utiliser le théorème des restes chinois, les nombres sont premiers entre eux, donc $d = 1$. De plus, il nous fait gagner une étape car il nous renvoie l'inverse de a modulo b. $x_0 \leftarrow 1$ $x_1 \leftarrow 0$ $y_0 \leftarrow 0$ $y_1 \leftarrow 1$ $s \leftarrow 1$ $d \leftarrow b$ $b \neq 0$ (q,r) $\leftarrow \text{divmod}(a, b)$ /* q est le quotient et r le reste de la division euclidienne de a par b */ $(a, b) \leftarrow (b, r)$ $(x, y) \leftarrow (x_1, y_1)$ $(r_1, y_1) \leftarrow (q \times s_1 + x_0, q \times y_1 + y_0)$ $(x_0, y_0) \leftarrow (x, y)$ $s \leftarrow -s$ $s \times x_0 + ((1 - s) \div 2) \times d$

2 Brute force

Il existe plusieurs algorithmes de brut force qui permettent de résoudre les erreurs tel que essayer tout les cas possibles. Mais cette algorithme est trop lourd à écrire. Nous avons donc fait des algorithmes moins lourds. Nous vous présentons 2 algorithmes dans cette section.

2.1 Notre premier algo version naïve

Tout d'abord, nous avons fait une fonction qui nous montre si il y a une erreur ou pas dans nos listes, car avant de vouloir trouver les erreurs, il faut savoir si il y en a une. Nous avons donc fait un petit algorithme qui enlève, chacun leurs tours, un élément des listes ; c'est à dire qu'on retire le premier élément des 2 listes, puis on retire que le deuxième élément, puis le troisième et ainsi de suite. Nous appliquons alors le théorème des restes chinois à chaque itération, et on met les résultats dans une liste auxiliaire. Si il n'y a pas d'erreur, lorsqu'on retourne cette liste auxiliaire, on devrait avoir la même valeur partout, si les valeurs de la liste sont différentes, nous avons une erreur.

Soit $P = [p_1, p_2, \dots, p_k]$ et $N = [n_1, n_2, \dots, n_k]$

On note $P_i = [p_1, \dots, p_i - 1, p_i + 1, p_k]$ et $N_i = [n_1, \dots, n_i - 1, n_i + 1, \dots, n_k]$

On note x_i le résultat théorème des restes chinois appliqué à P_i et N_i

Puis on retourne $L = [x_1, \dots, x_k]$

Si $x_1 = x_2 = \dots = x_k$ alors il n'y a pas d'erreur.

Voici un petit exemple pour bien comprendre cette fonction.

on pose $P=[2,3,5,7,11,13,17]$ et $N=[1, 1, 1, 5, 6, 9, 10]$

si on applique le théorème des restes chinois à ces 2 listes, on trouve 510510.

Pour vérifier qu'il n'y ai pas d'erreur, on applique le théorème à $P1=[3,5,7,11,13,17]$

et $N1=[1, 1, 5, 6, 9, 10]$, puis à $P2=[2,5,7,11,13]$ et $N2=[1, 1, 5, 6, 9, 10]$ jus-

qu'à $P7=[2,3,5,7,11,13]$ et $N7=[1, 1, 1, 5, 6, 9]$

Comme il n'y a pas d'erreur, la fonction va retourner [61, 61, 61, 61, 61, 61, 61]

à présent, nous mettons une erreur dans les listes, on a $P=[2,3,5,7,11,13]$ et $N=[1,1,1,5,8,10]$. En appliquant la fonction, il sera retourner une liste [9721, 9721, 3715, 1141, 1531, 4811], où l'on remarque bien que chaque élément est différent, donc nous avons au plus une erreur. Dans cette fonction, on parcourt n fois la liste de n élément donc nous sommes en $O(n^2)$. Mais on fait n fois l'algo des restes chinois donc on est en $O(n^2)$. Cette algorithme est donc en $O(n^4)$

Pour créer notre première fonction de correction, nous nous sommes inspirées de la fonction ci dessus. On sait dit que si on enlève 2 élément des listes, nous pourrions trouver l'erreur. On s'explique ; On a vu que si on en-

levais 1 éléments des listes, le théorème des restes chinois marchait encore, et si on enlève 2 éléments, il marche encore. Donc, en enlevant le premier élément et tout les autres chacun leurs tours, et en appliquant le théorème à chaque fois, on retourne une liste. Puis, on réitère en enlevant le deuxième élément, et les autres chacun leurs tours, en re-appliquant les restes chinois à chaque fois, en mettant les résultats dans une nouvelle liste. Ainsi de suite jusqu'au dernier élément. Cette fonction retourne une liste de liste. Donc la liste où se trouve le même nombre partout, c'est que l'erreur est à la même position. faisons plutôt exemple.

Soit $P = [p_1, p_2, \dots, p_k]$ et $N = [n_1, n_2, \dots, n_k]$

On note $P_{ij} = [p_1, \dots, p_i - 1, p_i + 1, \dots, p_j - 1, p_j + 1, \dots, p_k]$ et $N_{ij} = [n_1, \dots, n_i - 1, n_i + 1, \dots, n_j - 1, n_j + 1, \dots, n_k]$

On note x_{ij} le résultat du théorème des restes chinois appliqués à P_{ij} et N_{ij}

Et on retourne $L = [[x_{11}, x_{12}, \dots, x_{1k}], [x_{21}, x_{22}, \dots, x_{2k}], \dots, [x_{k1}, x_{k2}, \dots, x_{kk}]]$

On a retourné une liste de liste. Si la k-ième sous-liste retourne toujours le même nombre alors cela signifie qu'il y a une erreur sur le k-ième élément.

Cette fonction est très performante comme vous avez pu le voir sur les exemples, cependant il y a quelque défaut. Premièrement, cette fonction ne marche que avec une erreur, on peut facilement changer la fonction pour trouver n erreur juste en rajoutant des boucles FOR. Deuxièmement, nous avons un problème de borne. Pour trouver la borne, il suffit de multiplier par les 2 derniers nombre premier de la liste.

Par exemple pour le nombre 30030, la borne est $30030/(11 \times 13)$ soit 210.

Enfin, nous avons un problème de complexité soit $O(n^3)$ et à chaque fois on fait le théorème des restes chinois qui est en $O(n^2)$ l'ensemble des entiers de la zone d'encodage.

2.2 brute force de hamming

Notre deuxième algorithme brut force consiste à utiliser la distance de hamming. Soit $x = x_1x_2...x_n$ et $y = y_1y_2...y_n$, la distance de Hamming c'est $d_H(x, y) = Card\{i | x_i \neq y_i\}$

Et dans cette algorithme nous utilisons la distance de hamming. Nous laissons le choix au lecteur d'aller voir l'algorithme de la distance de hamming en annexe.

3 Les fractions continues

3.1 Principe

Comme vu précédemment la méthode brute force est efficace sur des petits cas de correction mais sur de cas plus grand la complexité explose et il est très difficile d'obtenir des résultats en un temps raisonnable.

Il faut donc trouver un moyen plus rapide et optimal. Donc comprenons comment intervient l'erreur sur le n-uplet du système d'équation.

Soit m le message envoyé et y le message et l'erreur dans le même message. si l'on fait la différence de $m-y$ on obtient e l'erreur. Maintenant cette erreur ce représente par un uplet en supposant par soucis de représentation que les erreurs se trouve au début de l'uplet associé $e = [e_1, e_2, 0, 0, \dots, 0]$ cette erreur est donc un multiple de tout les p_i nombres premiers pour $i > 2$ et 2 reste non nul sur les positions erroné.

Donc $y = m + e$ donc si l'on divise y par N notre entier on a $\frac{y}{N} = \frac{m}{N} + \frac{e}{N}$ mais $\frac{e}{N}$ peut se simplifier car hormis les positions erronés 2 dans notre cas e est un multiple de tout les autres p_i qui composent N

Ainsi on a $\frac{y}{N} = \frac{m}{N} + \frac{e}{N} = \frac{m}{N} + \frac{e_1 * e_2}{p_1 * p_2} < \frac{B}{N}$ B étant la borne choisie.

Finalement $\frac{m}{N} = \frac{y}{N} - \frac{e_1 * e_2}{p_1 * p_2}$

Or souvenons-nous que si z est un réel positif et que la fraction rationnelle $\frac{p}{q}$ vérifie $\left| z - \frac{p}{q} \right| < \frac{1}{2q^2}$. Alors $\frac{p}{q}$ est une fraction réduite de z .

Appliquer à notre problème il suffit donc de calculé toutes les réduites de $\frac{y}{N}$ jusqu'à que le denominateur dépasse le produit des 2 derniers nombres premiers du système.

Maintenant parlons de la complexité de cette méthode. On parcourt la liste des derniers nombres premiers du système, pour calculer une borne à ne pas dépasser au denominateur de la fraction cela réduit drastiquement les itérations à faire. Une fois cette borne déterminé on calcule les réduites de $\frac{y}{n}$ sans que le denominateur qui sert d'indication sur la position des erreurs ne dépasse la borne. A chaque tour de boucle on calcule une réduite qui coute l'ordre de la réduite fois le coût d'une division euclidienne, l'algorithme permettant de calculé une réduite étant une version modifié de l'algorithme d'Euclide arrêté à l'ordre de la fraction réduite voulue. Donc finalement pour calculer toutes les réduites qui mettent en évidence les potentiels position erronés le coût est relativement faible car dépend de la taille du système lineairement donc relativement faible et rapide. Coût que l'on a constaté grâce à nos expériences et test.

3.2 Résumé d'expériences et test

Au début de nos tests nous avons pratiqué avec des petits cas de 6 à 7 nombre premier et cette méthode se révéla très efficace. Mais pour tester des cas avec plus d'erreurs nous avons eu besoin de système plus grand nous avons donc directement étendu notre système à 51 nombres premiers. Déjà N est supérieur à 10^{80} , nous nous sommes restreint à 10^{27} comme espace d'encodage pour tester les cas avec plusieurs erreurs. Même si théoriquement nous pourrions résoudre un cas à 15 erreurs en pratique le système est trop erroné pour pouvoir le résoudre. Concrètement nous arrivés à corriger jusqu'à 7 erreur en encodant jusqu'à 10^{27}