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 conaissance de ce théorème, puis dans un segond temps, nous l'avons corrigé grace à plusieurs méthode, tel que du brut force ou une méthode de fractions continues.

Table des matières

Nous tenons tout d'abord à remercier Mr CASTAGNOS Guillaume d'avoir était 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 n1,...,nk des entiers premiers 2 à 2. Pour tout a1,...,ak, il existe un entier x tel que :
$x \equiv a1 \ (mod \ n1)$
$x \equiv a2 \ (mod \ n2)$
$x_k \equiv ak \ (mod \ nk)$
Nous pouvons démontrer ce théorème de la façon suivante :

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 répartis pour les 3 enfants et leurs parents, il en reste (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 :
$x \equiv 2 \ (mod \ 3)$
$x \equiv 3 \ (mod \ 5)$
$x_k \equiv 2 \ (mod \ 7)$
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.

1.2 notre algoritme

L'algotitme du théorème des restes chinois étant un peu lourd, nous allons montrer l'algoritme comme suivant avant de voir plus en détailles les algos en python. On a ici :
Soit n_i le i-ème termes de la liste des modulos, et on note
$N_i = \frac{n}{n_i} = n_1 n_2 ... n_i - 1 n_i + 1 ... n_k$
on a donc N_i et n_i qui sont premier entre eux.
Il faut alors faire l'algoritme d'euclide étendu sur N_i et n_i, se qui nous donne :
$1 = N_i u_i + n_i v_i$ où $u_i, v_i \in Z$
On pose donc $e_i = u_i N_i$ avec $e_i \equiv 1 \ (mod \ n_i)$ et $e_i \equiv 0 \ (mod \ e_j)$ avec $i \neq j$
On trouve donc une solution qui est $x = \sum_{i=1}^{k} a_i e_i$
Dans un premier temps, il a fallu faire l'algorithmes d'Euclide étendu, qui prend en paramètres a et b, deux entiers, et qui renvois l'inverse de a modulo b.

-PI la liste des nombres premier
-XI la liste des modulos
- on a posé a prend la valeur 1 car a sera après le produit des éléments de XI

2 Brute force

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 enlèves, 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 a chaque itération, et on mets le résultats dans une liste auxiliaire. Si il n'y a pas d'erreur, lorsqu'on retourne cette liste auxiliaire, on devrais avoir la même valeur partout, si les valeurs de la liste sont différentes, nous avons une erreur.
Voici un petit exemple pour bien comprendre cette fonction.
c'est écris différaments
Soit $P = [p_1, p_2, ..., p_k]$ et $N = [n_1, n_2, ..., n_k]$
On note $P_i = [p_1, ..., p_i - 1, p_i + 1, p_k]$ et $N_i = [n_1, ..., n_i - 1, n_i + 1, ..., n_k]$
On note x_i le résultat théorème des restes chinois appliqué à P_i et N_i
Puis on retourne $L = [x_1, ..., x_k]$
Si $x_1 = x_2 = ... = x_l$ alors il n'y a pas d'erreur.
on pose P=[2,3,5,7,11,13,17] et N=[1, 1, 1, 5, 6, 9, 10]
si on applique le théorèmes des restes chinois à ces 2 listes, on trouve 510510.
Pour vérifier qu'il ny 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=[2,5,7,11,13,17] et N2=[1, 1, 5, 6, 9, 10] jusqu'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=[2,3,5,7,11,13,17] et N=[1,1,1,5,8,10]. En appliquant la fonction, il sera retourner une liste [...], où l'on remarque bien que chaque élément est différrent, donc nous avons au plus une erreur. Dans cette fonction, on parcours n fois la liste de n élément donc nous somme en $O(n^2)$. Mais on fait n fois l'algo des restes chinois donc on est en $O()$

Pour créer notre première fonction de correction, nous nous sommes inspirées de la fonction ci dessus. On sait dit que si on enlevé 2 élément des listes, nous pourrions trouver l'erreur. On s'explique ; On a vu que si on enlevais 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 appliquantle théorème à chaque fois, on retourne une liste. Puis, on réitére 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 meme position. faisont plutôt exemple.

2.2 brute force de hamming

3 Les fractions continues

```
1  xs0 ← 1 ;
2  xs1 ← 0 ;
3  ys0 ← 0 ;
4  ys1 ← 1 ;
5  s ← 1 ;
6  d ← b ;
7  while b≠ 0 do
8  |  (q, r) ← divmod(a, b) ;
9  |  (a, b) ← (b, r) ;
10 |  (x, y) ← (xs1, ys1) ;
11 |  (rs1, ys1) ← (q × s1 + xs0, q × ys1 + ys0) ;
12 |  (xs0, ys0) ← (x, y) ;
13 |  s ← −s ;
14 end
15 return s×xs0 + ((1 − s) ÷ 2) × d
```