Algorithmes de factorisation de polynômes

Victor CASTIEL, Ivo DOULIERY et Victor GAUTHIER

1er juillet 2021

Table des matières

Introduction **2**

1 Étude des polynômes sur un corps fini **3**
1.1 Corps fini et premières operations 3
1.2 Étude des polynômes 4

2 Algorithme de factorisation sans carré **6**
2.1 Polynome sans carré 6
2.2 Algorithme naïf 6
2.3 Algorithme de Yun 7

3 Implémentation **10**

Conclusion **12**

Introduction

Le but de notre projet est de travailler sur différents algorithmes de factorisations de polynômes (par exemple l'algorithme de Yun). Pour cela nous avons étudié les polynômes sur un corps fini. Durant notre travail, nous avons commencé par créer des fonctions de bases pour les entiers modulaires (addition, soustraction, multiplication, etc...) puis quelques fonctions plus avancées d'arithmétique($PGCD$, Euclide étendu,...). Ensuite nous avons fait exactement la même chose mais avec les polynômes à coefficients dans un tel corps. Après avoir fait cela, nous nous sommes donc penchés sur le point important de notre sujet : les algorithmes de factorisation. Nous avons surtout travaillé sur les algorithmes prenant en comptent seulement les polynômes sans carré (algorithme naïf et de Yun).

1 Étude des polynômes sur un corps fini

1.1 Corps fini et premières operations

Un corps est un anneau $(K, +, .)$ avec unité, non nul, où tous les éléments différents de 0 sont inversibles pour la multiplication. L'ensemble $\mathbf{Z}/n\mathbf{Z}$ avec n nombre premier est donc un corps car les élements différents de 0 ont un inverse dans cet ensemble (grâce à l'algorithme d'Euclide étendue).
Avant de le montrer, nous allons nous intéresser à $\mathbf{Z}/n\mathbf{Z}$

$$\forall x \in \mathbf{Z}/n\mathbf{Z}, x[n] \equiv x + kn[n],\ \forall k \in \mathbf{Z}$$

Pour bien comprendre, nous ferons en sorte que $0 < x < n$
Intéressons-nous maintenant à l'algorithme d'Euclide étendue pour prouver que chaque entier modulaire a un inverse modulaire. Pour cela on regarde déjà l'identité de Bézout :

$$Soient\ a, b \in \mathbf{Z}. \exists u, v \in \mathbf{Z}\ tq\ au + bv = pgcd(a, b).$$

En prenant $a = x$ et $b = n$, comme n est un nombre premier, $pgcd(x, n) = 1$ et $nv[n] = 0$ donc il nous reste $xu[n] = 1$ et u est l'inverse modulaire de x.

Algorithme d'Euclide étendu :

$$Soit\ r_0 = a, r_1 = b.\ On\ veut\ u, v\ tq\ pgcd(r_0, r_1) = ur_0 + vr_1$$

$$r_0 = q_1 r_1 + r_2 \qquad r_2 = u_2 r_0 + v_2 r_1 \qquad u_2 = 1, v_2 = -q_1$$

$$r_1 = q_2 r_2 + r_3 \qquad r_3 = u_3 r_0 + v_3 r_1 \qquad u_3 = -q_2, v_3 = 1 + q_1 q_2$$

$$...$$

$$r_{l-2} = q_{l-1} r_{l-1} + r_l \qquad r_l = u_l r_0 + v_l r_1 \qquad u_l = ..., v_l = ...$$

On s'arrête lorsque $r = 0$, on a ensuite :

$$pgcd(a, b) = r_l, u = u_l, v = v_l.$$

De manière informatique, on a :

```
Entree : a,b deux entiers strictement positifs
Sortie : r0,u0,v0 donnant la relation de Bézout (r0=PGCD(a,b), u0=u, v0=v)
r0=a
r1=b
u0=1
v0=0
u1=0
v1=1
Tant que r1!= 0 :
        q = r0/r1
        r2 = r0
        u2 = u0
        v2 = v0
        r0 = r1
        u0 = u1
        v0 = v1
        r1 = r2 - q*r1
        u1 = u2 - q*u1
        v1 = v2 - q*v1
Fin tant que
```

C'est donc grâce à la relation de Bézout et cet algorithme que nous pouvons trouver l'inverse d'un entier modulo un nombre premier.

1.2 Étude des polynômes

Un polynôme P sur un corps $\mathbf{K}$ est une expression algébrique consituée par une somme algébrique de monômes. Ils sont de la forme :

$$\forall X \in \mathbf{K}, P(X) = a_n X^n + ... + a_1 X + a_0$$

Ici comme corps $\mathbf{K}$ on prendra donc le corps $\mathbf{Z}/n\mathbf{Z}$ avec n premier.

Notre but par rapport est de factoriser un tel polynôme grâce à des algorithmes que nous allons étudier juste après, mais pour cela nous avons besoin d'opérations arithmétiques et plus précisément la division euclidienne et du $PGCD$ de deux polynômes :

-division euclidienne :

$$\forall P_1, P_2 \in \mathbf{K}[\mathbf{X}], P_2 \neq 0 \Rightarrow \exists!(Q, R) \in \mathbf{K^2}[\mathbf{X}], tq P_1 = P_2 Q + R$$

$$Avec\, R = 0\, ou\, deg(R) < deg(P_2)$$

Pour trouver les polynômes Q et R, on va procéder de manière analogue à la division euclidienne sur $\mathbf{Z}$, en élliminant les termes de plus hauts degrés. Exemple avec :

$$P_1(X) = 3X^2 + 2X + 1,\ P_2(X) = X + 2\, et\, Q(X) = aX + b$$

$$P_1(X) = 3X^2 + 2X + 1 = 3X(X + 2) - 4X + 1\, donc\, a = 3$$
$$P_1(X) = 3X^2 + 2X + 1 = (3X - 4)(X + 2) + 9\, donc\, b = -4$$
$$On\, a\, donc\, Q = 3X - 4\, et\, R = 9$$

-$PGCD$:
La manière d'obtenir le $PGCD$ de deux polynômes se fait à partir du moment où l'on sait faire une division euclidienne. Il faut faire la division euclidienne entre les deux polynômes,si le reste n'est pas nul, on renvoie le diviseur, si le reste est nul, on renvoie la division euclidienne du diviseur et du reste. Il tient en un algorithme (algorithme d'Euclide) :

$$faire\, la\, division\, euclidienne\, de\, P_1\, P_2,\ R = 0 \Rightarrow PGCD(P_1, P_2) = Q$$

$$R \neq 0 \Rightarrow PGCD(P_1, P_2) = PGCD(P_2, R)$$

2 Algorithme de factorisation sans carré

2.1 Polynome sans carré

Un polynôme sans carré est un polynôme qui ne possède aucune racine multiple. Une factorisation sans carré est une factorisation qui rend un polynôme avec des facteurs sans carrés, en nous donnant leur multiplicité.

Par exemple le polynôme

$$P(X) = X^2 + X - 2$$

est un polynôme sans carré car il peut s'écrire sous la forme

$$(X - 1)(X + 2)$$

2.2 Algorithme naïf

Soit un polynôme P à valeurs dans $\mathbf{Z}/n\mathbf{Z}$. L'algorithme naïf consiste à tester tous les entiers entre 0 et n-1 pour voir s' ils sont racines de P, puis s' ils le sont, le revérifier pour connaître leur multiplicité. Il va nous rendre un polynôme sans carré, qu'il faudra modifier en fonction de la multiplicité des racines. L'algorithme se déroule comme suit :

```
Entree : p un polynôme de modulo n de degrée d
Sortie : res un tableau de polynome, m un tableau d'entier
res=[NULL, NULL,...] initialisé de taille d
m=[ ] initialisé de taille d
p2=p
compteur=0
i=0

Tant que i <=n :
      Tant que p2(i)=0 : // tant que i est racine du polynôme
            res[compteur]=X+n-i
            m[compteur]=i+n-i
            p2=p2/(X+n-i)
            compteur++
      Fin Tant que
      i++
Fin Tant que
```

On regarde si le nouveau polynôme est nul en i, on répète l'opération jusqu'à ce qu'il ne soit plus égal à 0. Le nombre de fois où l'on réitère l'opération est la multiplicité de cette racine. Á la fin on a donc les racines et la multiplicité. Le seul problème de cet algorithme c'est que pour des grandes valeurs, il est très lent au vu de sa complexité.

Exemple :

$$Soit\ P(X) = X^3 - 2X^2 + X,\ \forall X \in \mathbf{Z}/7\mathbf{Z}$$

$$P(0) = 0,\ P(X)/X = X^2 - 2X + 1$$

$$En\ 0\ on\ a\ :\ 0^2 - 2 * 0 + 1 = 1\ donc\ 0\ racine\ simple$$

$$P(1) = 0, P_1(X) = P(X)/(X-1) = X^2-X, P_1(1) = 0, P_2(X) = P_1(X)/(X-1) = X$$

$$P_2(1) = 1,\ donc\ 1\ racine\ double$$

$$\forall i \in \{2, ..., 6\},\ P(i) \neq 0$$

$$Donc\ P(X) = X(X-1)^2,\ \forall X \in \mathbf{Z}/7\mathbf{Z}$$

2.3 Algorithme de Yun

Soit P un polynôme non nul à coefficients dans $\mathbf{Z}/n\mathbf{Z}$. On souhaite factoriser ce polynôme de la forme :

$$P = P_1 P_2^2 P_3^3 ... P_n^n$$

où les P_i sont premiers entre eux et sans carré. On pose :

$$P_0 = P_2^1 P_3^2 ... P_n^{n-1}$$

$$P/P_0 = P_1 P_2 P_3 ... P_n$$

On sait aussi que :

$$P' = \sum_{i=1}^{n} iP_i' P_1 P_2^2 ... P_{i-1}^{i-1} P_i^{i-1} P_{i+1}^{i+1} ... P_n^n$$

$$P'/P_0 = \sum_{i=1}^{n} iP_i' P_1 P_2 ... P_{i-1} P_{i+1} ... P_n$$

Comme les P_i sont premiers entre eux, $PGCD(P/P_0, P'/P_0) = 1$ donc $PGDC(P, P') = P_0$. En effet, $PGCD(P_i, P_j) = 1$ car les P_i sont premiers entre eux. Donc $PGCD(P_i, P_i' P_1 P_2 ... P_{i-1} P_{i+1} ... P_n) = 1$ ce qui entraîne que $PGCD(P, P_i' P_1 P_2 ... P_{i-1} P_{i+1} ... P_n) = 1$.
On pose maintenant les bases de l'algorithme :

$$P_0 = PGCD(P, P');\ B_1 = P/P_0;\ C_1 = P'/P_0;\ D_1 = C_1 - B_1';\ i = 1$$

$$P_i = PGCD(B_i, D_i);\ B_{i+1} = B_i/P_i;\ C_{i+1} = D_i/P_i;\ i = i + 1;\ D_i = C_i - B_i'$$

On arrête l'algorithme lorsque $b = 1$ et on récupère donc au fur et à mesure tous les P_i.

On voit que $P_i = PGCD(B_i, D_i)$ car :

$$\forall i \in [1, n],\ B_i = P_i P_{i+1} ... P_n,\ B_i' = \sum_{j=i}^{n} P_j' P_i .. P_{j-1} P_{j+1} ... P_n$$

$$C_i = \sum_{j=i}^{n} (j - i + 1) P_j' P_i ... P_{j-1} P_{j+1} ... P_n \, et \, D_i = \sum_{j=i+1}^{n} (j - i) P_j' P_i .. P_{j-1} P_{j+1} ... P_n$$

On voit bien que dans la somme de D_i, P_i est présent dans chaque facteur. Donc comme précedemment, $PGCD(B_i, D_i) = P_i$, car aucun autre P_j, $j > i$ ne divise B_i et les P_j sont premiers entre eux.

De manière plus informatique, l'algorithme se déroule comme suit :

```
Entree : p un polynôme de degrée d et de modulo n
Sortie : res un tableau de polynôme
res=[NULL,NULL,..] de taille d
a=PGDC(p,p')
```

```
b=p/a
c=p'/a
d=c-b'
i=1
res[0]=a
Tant que b !=1 :
      a=PGCD(b,d)
      b=b/a
      c=d/a
      d=c-b'
      res[i]=a
      i++
Fin tant que
```

3 Implémentation

Au niveau de l'implémentation, on a du faire plusieurs changement sur notre approche de le factorisation des polynômes. Nous avons en premier pensé à représenter un polynôme sous forme d'un tableau de réel. Mais le polynôme $x^2 - 2$, qui a pour racine $\sqrt{2}$, nous a vite montré qu'il fallait changer de structure pour nos polynômes car $\sqrt{2}$ allait être compliqué à trouver même par dichotomie.

Nous avons donc cherché un corps qui nous permettrait d'avoir des racines entières afin de pouvoir exprimer notre polynôme comme un tableau d'entier. Notre objectif étant de factoriser un polynôme, il a donc fallu utiliser une succession de division euclidienne de polynômes mais on a vite rencontré un problème. En effet, si on divise x^2 par $2x$, le quotient aura un coefficient dominant de $1/2$, or $1/2$ n'est pas un entier et donc le polynôme quotient devenait nul car le $1/2$ était casté en 0.

On a donc fini par utiliser le corps $\mathbf{Z}/n\mathbf{Z}$, car dans un premier temps cela nous permettait de réduire les coefficients dans un intervalle $[0, modulo - 1]$. Puis dans un second temps cela nous permettait de calculer l'inverse modulaire d'un entier qui est lui aussi un entier, donc en si on reprend notre exemple avec x^2 par $2x$ en modulo 5 on a le coefficient dominant qui sera $1 * inverseModulaire(2)$, soit $1 * 3 = 3$. Donc notre quotient ne sera pas le polynôme nul à chaque fois.

On avait donc le moyen de faire des divisions euclidienne entre deux polynômes de manière relativement propre. On avait absolument besoin de cela pour construire notre algorithme naïf.

Algorithme naïf :

Notre approche était que sachant que nous étions sur un corps d'entier modulo un nombre premier modulo, nous savions que l'ensemble des valeurs que pouvaient prendre les racine R du polynôme P était compris dans l'intervalle $[0, modulo - 1]$. En trouvant une première racine, on savait que

$$P = Q * (X + modulo - R)$$

avec Q le quotient de $P/(X + modulo - R)$.

Il nous reste à trouver sa multiplicité, donc on réitère l'opération avec P qui devient Q jusqu'à ce que R ne soit plus racine. On passe à la potentielle racine suivante et ainsi de suite jusqu'à trouver toutes les racines et leur multiplicité tout en sauvegardant les polynômes $(X + modulo - R)$ avec R la racine, dans un tableau de polynômes que nous retournons à la fin de la boucle. Cas échéant : si la somme des multiplicité des racines n'est pas égal au degré du polynôme initial, alors c'est que le polynôme P n'était pas scindable (cela arrive lorsque le modulo n'est pas premier).

Algorithme de Yun :

L'algorithme de Yun, permet de décomposer un polynôme P en un produit de facteurs sans carré $P = A1^1A2^2A3^3\ldots$ avec tous les Ai premiers entre eux.
Nous avons donc étudié l'algorithme afin de pouvoir l'implémenter en C.
On a imaginé une fonction qui retourne un tableau de polynômes qui contient les Ai.
Malheureusement, nous avons rencontré quelques difficultés.
Premièrement, les polynômes qui étaient entrés dans le tableau étaient les bons mais à un facteur près, par exemple, on avait $3x^2 + 4$ au lieu de $x^2 + 6$ en modulo 7. On a bien $3 * (x^2 + 6) = 3x^2 + 4$ en modulo 7. On a donc divisé les résultats par le coefficient dominant A grâce à l'inverse modulaire de A au modulo en question.
Deuxièmement, nous avions eu un problème au niveau de l'ordre des Ai dans le tableau. En effet, si un ou plusieurs des Ai sont égaux à 1, alors notre programme mettait les Ai à la suite sans prendre en compte l'écart qu'il devait y avoir entre eux dans le tableau en fonction de leur multiplicité. On a une solution qui serait de créer une fonction qui calcule la multiplicité de chaque racine afin de les placer au bon indice dans le tableau, mais la complexité serait beaucoup trop grande. On va donc chercher une solution pour le jour de la soutenance.

Conclusion

Avec un peu de recul, ce projet était très intéressant au travers du nombre conséquent d'échecs auxquels nous avons dû faire face. En effet nous avons dû recommencer plusieurs fois notre code de zéro afin de contrer certains problèmes, comme changer de structure pour les polynômes, recommencer les fonctions pour les adapter au corps $\mathbf{Z}/n\mathbf{Z}$, ...
Conscient du fait que nos algorithmes ne vont pas révolutionner le monde de l'algorithmique, nous avons pris beaucoup de plaisir à nous diviser les tâches en fonction des points forts de chacun, à travailler en groupe et à goûter, même si ce n'était qu'un en-cas, au monde de la recherche.

Lien github : https ://github.com/ivodouliery/Projet-ete.git