Egison による因数分解

梅崎直也@unaoya

2019 年 4 月 3 日

多項式の因数分解プログラムについて

今回は有限体上での多項式の因数分解通常、中学校などでやるような整数の範囲での因数分解は、今回の有限体上での分解に加え、Hensel 持ち上げというを組み合わせることで実装可能ですが、今回はそこまでの時間がなかったため

有限体上での因数分解のアルゴリズムは以下の三つのステップがあります。

これらについて、簡単にアルゴリズムの説明と数学的背景、Egison による実装を紹介します。

1 有限体

最初に準備として、有限体とは何か？また有限体上の多項式の演算について紹介します。有限体はガロア体などとも呼ばれる

有限体は要素の数が必ず素数の冪乗 p^e になります。また要素の数を決めると、一意的に決定されます。

今回は要素の数が素数の場合を使います。(付録で要素の数が素数でない場合を紹介します。)

有限体の演算を実装しましょう。

体なので、四則演算を定義します。Egison では modulo という関数があるので、それを使えばよいです。(modulo)

ただし割り算には注意が必要です。逆元を探す操作を

例えば Fermat の小定理などもこのように確かめることができます。

次に有限体を係数にもつ多項式の演算を実装していきます。こちらも足し算、引き算、掛け算とあまりのある割り算を実装します。また後で使うので、微分、公約数、

2 $\mathbb{F}_p[x]$ での因数分解

例えば $p = 3$ では

$$
\begin{aligned}
x^3 + x^2 + x + 1 &= (x^2 + 1)(x + 1) \\
x^2 + 2 &= x^2 + 3x + 2 \\
&= (x + 1)(x + 2) \\
x^3 + 1 &= x^3 + 3x^2 + 3x + 1 \\
&= (x + 1)^3
\end{aligned}
$$

3 $\mathbb{F}_p[x]$ での演算

多項式の四則、ベキ乗、割り算、gcd、微分などを係数 $\bmod p$ する。

```
(define $coef-map
  (lambda [$f $P $x]
    (sum' (map2 2#(*' %1 (**' x %2)) (map f (coefficients P x)) nats0))))

;operation on Fp[x]
(define $coef-mod
  (lambda [$P]
    (coef-map 1#(modulo %1 p) P x)))

(define $p.b.+
  (lambda [%x %y] (coef-mod (+ x y))))

(define $p.+
  (cambda $xs (foldl p.b.+ (car xs) (cdr xs))))

(define $p.b.-
  (lambda [%x %y] (coef-mod (- x y))))

(define $p.-
  (cambda $xs (foldl p.b.- (car xs) (cdr xs))))

(define $p.b.*
  (lambda [%x %y] (coef-mod (* x y))))

(define $p.*
  (cambda $xs (foldl p.b.* (car xs) (cdr xs))))

(define $p.inv
  (lambda [%x]
    (car (filter (lambda [$a] (eq? (p.* a x) 1)) (take p nats)))))

(define $p.b./
  (lambda [%x %y] (p.* x (p.inv y))))

(define $p./
  (cambda $xs (foldl p.b./ (car xs) (cdr xs))))

(define $p.**
  (lambda [$x $n]
    (match n integer
      {[,0 1]
       [_ (p.* x (p.** x (- n 1)))]})))


(define $p.L./
  (lambda [$xs $ys]
    (if (lt? (length xs) (length ys))
      [{} xs]
      (match [ys xs] [(list math-expr) (list math-expr)]
        {
         [[<cons $y $yrs> <cons $x $xrs>]
```

```
         (let {[[$zs $rs] (p.L./ {@(map2 p.- (take (length yrs) xrs) (map (p.*\
 $ (p./ x y)) yrs))
                                 @(drop (length yrs) xrs)} ys)]}
           [{(p./ x y) @zs} rs])]
        })))))

(define $p.P./
  (lambda [$fx $gx $x]
    (let* {[$as (reverse (coefficients fx x))]
           [$bs (reverse (coefficients gx x))]
           [[$zs $rs] (p.L./ as bs)]}
      [(sum' (map2 2#(*' %1 (**' x %2)) (reverse zs) nats0))
       (sum' (map2 2#(*' %1 (**' x %2)) (reverse rs) nats0))])))

(define $monic
  (lambda [$f $x]
    (/ f (rac (coefficients f x)))))

(define $p.monic
  (lambda [$f $x]
    (p./ f (rac (coefficients f x)))))

;gcd(f,0)=f
(define $gcd
  (lambda [$f $g $x]
    (if (eq? g 0) f
      (let {[$q (fst (P./ f g x))] [$r (snd (P./ f g x))]}
        (match r math-expr
          {[,0 (monic g x)]
           [_ (gcd g r x)]})))))

(define $p.gcd
  (lambda [$f $g $x]
    (if (eq? g 0) f
      (let {[$q (fst (p.P./ f g x))] [$r (snd (p.P./ f g x))]}
        (match r math-expr
          {[,0 (p.monic g x)]
           [_ (p.gcd g r x)]})))))

(define $deg
  (lambda [$f $x]
    (length (coefficients f x))))

(define $p. ∂/∂
  (lambda [$f $x]
    (coef-mod (∂/∂ f x))))

;rewrite p-th power (x^p -> x)
(define $rewrite-rule-p-power
  (lambda [$term]
    (match term math-expr
      {[(* $a ,x^(& ?(divisor? $ p) $k) $r)
        (*' a r (**' x (/ k p)))]
       [_ term]})))

(define $p-inv (map-terms rewrite-rule-p-power $))
```

4 因数分解アルゴリズムの実装

次の三段階で因数分解を行うアルゴリズム

参考:Wikipedia, Factorization of polynomials over finite fields

1. $f(x)$ を重複度ごとに分解、Square-free factorization
2. 既約因子の次数ごとに分解、Distict-degree factorization
3. 次数成分ごとに既約分解、Cantor-Zassenhaus algorithm

5 Square-free factorization

$$f(x) = f_1(x)f_2(x)^2f_3(x)^3\cdots f_n(x)^n$$

と各因子 $f_i(x)$ は重複因子を持たず、それぞれ互いに素であるように分解

例えば $f(x) = x^4+ = (x^2+2x)(x^2+2x+1)$ と分解します。ここで、各因子は必ずしも既約式には分解していないことに注意してください。

$$\begin{aligned} f(x) &= x^{11}+2x^9+2x^8+x^6+x^5+2x^3+2x^2+1 \in \mathbb{F}_3[x] \\ &= (x+1)(x^2+1)^3(x+2)^4 \end{aligned}$$

6 SFF 実装

f' と f の共通約数が重複因子であることを用いる。例えば $f(x) = g(x)^2h(x)$ となっているとすると、積の微分公式を使うと $f'(x) = (g(x)^2)'h(x) + g(x)^2h'(x) = 2g'(x)g(x)h(x) + g(x)^2h'(x) = g(x)(2g'(x)h(x) + g(x)h'(x)$ となり、$g(x)$ が共通因子としてくくり出せます。つまり、重複因子があれば $f(x)$ と $f'(x)$ の公約数になるということです。

上の例で言えば、

ただし標数 $p > 0$ のときは $(x^p)' = 0$ なので注意が必要。

```
(load-file "fp-operation.egi")

;square free factorization in ch 0
;Yun's algorithm
(define $Yun'
  (lambda [$f $g $x]
    (let* {[$a (gcd f g x)]
           [$b (fst (P./ f a x))]
           [$c (fst (P./ g a x))]
           [$d (- c (∂/∂ b x))]}
      (match b math-expr
        {[,1 {a}]
         [_ (cons a (Yun' b d x))]}))))
```

```
(define $SFF0
  (lambda [$f $x]
    (cdr (Yun' f (∂/∂ f x) x))))

;square free factorizatin in ch p
(define $step
  (lambda [$i $w $c $j]
    (let* {[$y (p.gcd w c x)]
           [$fac (fst (p.P./ w y x))]
           [$d (* i (** p j))]}
      (match y math-expr
        {[,1 {[(fst (p.P./ c y x))] [fac d]}]
         [_ (append (step (+ i 1) y (fst (p.P./ c y x)) j) {[fac d]})]}))))

(define $SFFp'
  (lambda [$f $x $j]
    (let* {[$g (p. ∂/∂ f x)]
           [$c (p.gcd f g x)]
           [$w (fst (p.P./ f c x))]
           [$R (step 1 w c j)]}
      (match (car R) math-expr
        {[,1 (cdr R)]
         [_ (append (cdr R) (SFFp' (p-inv (car R)) x (+ j 1)))]}))))

(define $fstnot1? (lambda [$t] (not (eq? (fst t) 1))))
(define $SFFp (lambda [$f $x] (filter fstnot1? (SFFp' f x 0))))
```

7 Distict-degree factorization

ここでは、多項式を各因子が同じ次数の既約因子の積となるように分解します。

$$f(x) = f_1(x)f_2(x)\cdots f_d(x)$$

で $f_i(x)$ の既約因子が i 次式であるように分解する。

これも各因子を分解していないことに注意。

$$x^4 + 2 = (x^2 + 2)(x^2 + 1)$$

8 DDF 実装

$\mathbb{F}_{q^i}$ の乗法群が位数 $q^i - 1$ の有限群であることから、$x^{q^i} - x$ との公約数を取ればよい。

```
(load-file"fp-operation.egi")

;distinct degree factorization

(define $DDF'
  (lambda [$f $i $x]
```

```
    (let* {[$g (p.gcd (- (** x (** p i)) x) f x)]
           [$q (p.monic (fst (P./ f g x)) x)]}
      (match q math-expr
        {[,1 {[f i]}]
         [_ (cons [g i] (DDF' q (+ i 1) x))]})))))


;f to be square-free polynomial
(define $DDF
  (lambda [$f $x] (DDF' f 1 x)))
```

9 Cantor-Zassenhaus

重複因子を持たず、既約成分が全て d 次である多項式を因数分解する。

```
(load-file "fp-operation.egi")

;f is the polynomial to factor
;x is its variable
;d is the degree of the factors of f
;b is a randomly chosen polynomial
(define $Cantor-Zassenhaus
  (lambda [$f $b $x $d]
    (let* {[$m (/ (- (** p d) 1) 2)]
           [$a (p.** b m)]}
      {(p.gcd f a x) (p.gcd f (p.+ a 1) x) (p.gcd f (p.- a 1) x)})))
```

10 今後の課題

1. 有理数係数での因数分解（Hensel 持ち上げ）
2. 代数的数の扱い（書き換え規則でできそう）
3. 積分
4. いちいち　mod p での演算を実装し直すのが面倒

11 参考文献

1. Wikipedia, Factorization of polynomials over finite fields,
2. https://github.com/unaoya/factorization