UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE MATEMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM MATEMÁTICA APLICADA

Aritmética de Corpos Finitos: Algoritmos para a Fatoração Polinomial

por

Ruth Noemi Noriega Sagastegui

Dissertação submetida como requisito parcial
para a obtenção do grau de
Mestre em Matemática Aplicada

Prof. Vilmar Trevisan
Orientador

Porto Alegre, Janeiro de 1996.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Noriega Sagastegui, Ruth Noemi

Aritmética de Corpos Finitos: Algoritmos para a Fatoração Polinomial / Ruth Noemi Noriega Sagastegui.— Porto Alegre: CPGMA da UFRGS, 1996.

99 p.: il.

Dissertação (mestrado)—Universidade Federal do Rio Grande do Sul, Instituto de Matemática, Curso de Pós-Graduação em Matemática Aplicada, Porto Alegre, 1996. Orientador: Vilmar Trevisan

Dissertação: Computação Algébrica e Algoritmos
Palavras Chaves: Algoritmos Algébricos, Fatoração Polinomial, Corpos Finitos, Aritmética

A meus queridos filhos
Ruthy, Rosita e Obidio Jr.

AGRADECIMENTOS

O primeiro que quero fazer é agradecer a Deus pela sua infinita bondade comigo.

Também gostaría de agradecer ao meu orientador Vilmar Trevisan, pelos acertados e valiosos conselhos neste trabalho.

Quisera também agradecer ao profesor Julio R. Claeyssen e a todos os professores do instituto, de quem aprendi idéias inovadoras, que serviram na minha vida profissional.

De forma especial quero agradecer ao profesor Alejandro Ortiz Fernández, pela confiança depositada em mim.

Finalmente quero agradecer a meu esposo, Obidio, pelos conselhos e críticas construtivas ao respeito deste trabalho.

Quero agradeçer a CAPES pela ajuda económica durante o desenvolvimento deste trabalho.

RESUMO

Este trabalho descreve algoritmos algébricos para computação em corpos de Galois $GF(q)$, com $q = p^n$, onde p é a característica do corpo, que pode ser arbitrariamente grande. Para fundamentar esse estudo é condensada e apresentada toda a ferramenta algébrica necessária. Os corpos finitos são caracterizados, é mostrado como construí-los e sua aritmética é analisada. Algoritmos determinísticos e probabilísticos são desenvolvidos para o cálculo de raízes polinomiais e a fatoração de polinômios sobre esses corpos. Este trabalho é materializado pela implementação de dois algoritmos, o de Cantor-Zassenhaus e o de Rabin, ambos implementados no Sistema de Computação Algébrica MAPLE V Release 3.

ABSTRACT

This work describes algebraic algorithms for computing in Galois Fields $GF(q)$, with $q = p^n$, where p is the characteristic of the field and may be arbitrarialy large. By justifying this work we give a colection of results about topics of Algebra. Deterministics and probabilistics algorithms are developed to compute polynomials roots and for polynomial factorization in $GF(q)$.This work is materialized by the implementation of two algorithms, Cantor-Zassenhaus's algorithm and Rabin's algorithm, both implemented in MAPLE V Release 3 Computer Algebra System.

SUMÁRIO

1 INTRODUÇÃO

Este trabalho tem como objetivo o estudo de corpos finitos. Além da sua caracterização também nos preocupamos com a sua estrutura e aritmética. Adicionalmente, dois problemas foram estudados com detalhe: A fatoração polinomial e o cálculo de raízes de polinômios em corpos finitos.

As motivações para este trabalho foram múltiplas. Além da utilidade como um módulo de algoritmos na Computação Algébrica, os corpos finitos representam uma estrutura apropriada para introduzir a idéia de algoritmos probabilísticos nestes corpos. Estudamos o algoritmo determinístico de Berlekamp [5] e os algoritmos probabilísticos de Cantor-Zassenhaus [15], de Rabin [28] e de Moenck [26], aprofundando naqueles que foram implementados. Os algoritmos de fatoração são importantes pelas aplicações dentro da própria Computação Algébrica, que incluia resolução de integrais simbólica, fatoração de polinômios inteiros. Além disso uma das motivações para a fatoração em corpos finitos são as aplicações em Teoria dos Códigos ([4]) e aplicações na teoria de comunicações .

Os algoritmos probabilísticos apresentados aqui são assim chamados porque fazem uso de polinômios aleatórios para a sua execução . Nesse sentido, o custo dos algoritmos são determinados baseados em uma distribução uniforme dos coeficientes polinomiais. Neste trabalho não nos preocupamos com esse importante e fascinante assunto que é a geração de números aleatórios. A referencia mais significativa para essa área é [22],cap.3.

Apresentamos uma coleção de resultados conhecidos que são a base para a caracterização de corpos finitos. Damos as provas dos teoremas que julgamos importantes.

Quanto ao tempo computacional dos algoritmos, fazemos as análises de custo somente daqueles que são importantes e necessárias para o nosso trabalho.

Ainda neste capítulo, damos uma descrição da computação Algébrica, e mostramos, usando exemplos do MAPLE V.3, as vantagens e desvantagens desta disciplina. Essa descrição se justifica pois Computação Algébrica é a área da Matemática na qual este trabalho se insere.

No capítulo 2, estudamos todos os pré-requisitos algébricos para a compreensão e fundamentação deste trabalho, em particular ao que nos leva ao desenvolvimento de corpos finitos.

No capítulo 3 caracterizamos os corpos finitos. Mostramos como construir tais corpos de Galois e como a aritmética é conduzida nesses corpos. Esse capítulo apresenta-se como definidor de nosso *ambiente de trabalho.*

No capítulo 4, descrevemos detalhadamente os algoritmos de Berlekamp e de Cantor-Zassenhaus, para a fatoração polinomial. A obtenção da matriz de Berlekamp utilizada em outros algoritmos é objeto de estudo pormenorizado.

No capítulo 5, nos referimos aos algoritmos de Moenck e de Rabin, para cálculo de raízes polinomiais. São dois algoritmos importantes que se adequam para mostrar a diferença entre algoritmos determinísticos e probabilísticos. Além disso, como aplicação do método de Rabin, introduzimos um outro algoritmo para a fatoração polinomial.

As considerações finais são apresentadas no último capítulo, e no anexo apresentamos uma implementação dos métodos de Rabin e Cantor-Zassenhaus para a fatoração polinomial em corpos finitos $GF(p)$, Também apresentamos uma implementação para o cálculo de raízes polinomiais e a sua aplicação para a fatoração polinomial.

A seguir faremos uma breve descrição do que é a Computação Algébrica, apresentamos alguns exemplos e discutimos algumas vantagens e limitações desta nova ferramenta tecnológica.

1.1 O que é a Computação Algébrica

Historicamente o verbo *computar* tem sido usado em relação a computação de números. A computação numérica não só está relacionada com as operações aritméticas básicas, mas também com cálculos mais sofisticados como funções matemáticas de valor numérico; achando raízes de polinômios e calculando autovalores numéricos de matrizes.

Para muitos cientistas matemáticos, computação e cálculo numérico tem-se convertido em sinônimos. Mas a Computação cietífica tem outra componente importante que chamamos de **Computação Algébrica ou Simbólica**. Em poucas palavras, isto pode ser definido como computação com símbolos representando objetos matemáticos. Estes símbolos podem representar números tais como inteiros, racionais, reais e complexos ou números algébricos, eles podem ser usados por objetos matemáticos como funções polinomiais e racionais, sistemas de equações, e ainda por estructuras algébricas como grupos, anéis e álgebras.

O adjetivo *simbólico* enfatiza que em muitos casos o último intento na solução de problemas matemáticos é expressar a resposta numa fórmula fechada ou achar uma aproximação simbólica.

Por *algébrico* entendemos que os cálculos são feitos exatamente de acordo com as regras da Álgebra. Por exemplo: Fatoração de polinômios, diferenciação, integração, expansão em séries de funções, soluções exatas de sistemas de equações e simplificação de expressões matemáticas.

Nos últimos 20 anos um grande progresso foi feito com respeito à base teórica dos algoritmos algébricos e simbólicos. Exemplos de todo esse progresso podem ser encontrados em [17], sendo que os mais significativos dizem respeito a fatoração polinomial, redução de equações é integração simbólica.

Devido a complexidade dos cálculos, geralmente envolvidas em computação algébrica, sempre que algum software tem de ser desenvolvido, é necessário um conjunto básico de operações (tais como simplificação , redução , e forma canónica, etc), o que implica na obtenção de um sistema de computação algébrica; um conjunto de rotinas, geralmente grandes com um objetivo específico ou de propósito geral. Enumeramos aqui alguns desses sistemas mais conhecidos.

1.2 Sistemas de Computação Algébrica

1. Sistemas de Propósito especial São usadas para resolver problemas em áreas específicas da Física e da Matemática. Alguns dos conhecidos são:

i) CAMAL (Mecânica Celestial)

ii) CAYLEY and GAP (Teoria de Grupos)

iii) CoCoA (Álgebra Comutativa)

iv) DELIA (Análise de Equações Diferenciais)

v) LIE (Teoria de Grupos de Lie).

vi) PARI (Teoria de Grupos)

vii) SCHOONSCHIP(física(energia))

viii) SHEEP and STENSOR (Relatividade Geral)

2. Sistema de Propósito Geral Dão a seus usuários uma grande variedade de estrutura de dados e funções matemáticas tratando de cobrir diferentes áreas de aplicação como seja possível. Entre elas temos:

i) AXIOM (1993, para sistemas UNIX)

ii) DERIVE

iii) MACSYMA (o mais antigo)

iv) MAPLE

v) MATHEMATICA

vi) REDUCE

1.3 Vantagens

A principal vantagem de um sistema de Computação Algébrica é a sua capacidade para fazer grandes cálculos algébricos, sem erros.

Exemplo.- Para solucionar

$$(\frac{\partial^2}{\partial x^2}(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+\frac{\partial^2}{\partial z^2})+n^2(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}))f=0$$

onde,

$$f=\frac{sin(\frac{nz\sqrt{x^2+y^2+z^2}}{\sqrt{y^2+z^2}})}{\sqrt{x^2+y^2+z^2}}$$

Usando MAPLE V Release 3, sobre uma máquina com 64 MB de memória principal mais 200 MB de espaço swap, leva 3.667 segundos.

A computação Algébrica, freqüentemente precede à computação numérica. As fórmulas matemáticas são primeiramente manipuladas para que no final sejam computadas numericamente, por esta razão é importante que um sistema de computação Algébrica tenha uma boa interface entre estes dois tipos de computação.

Usando um sistema de computação Algébrica, podemos nos concentrar na análise de um problema matemático, deixando os detalhes computacionais ao computador.

Uma outra vantagen da Computação Algébrica é a precesão arbitraria que existe nos sistemas. Isso quer dizer que podemos extrair a precisão que quiseremos, bastando para isso adequar certos parâmetros. Naturalmente a precisão fica restringida pela memória da máquina, mas, em principio, números inteiros como 2 e 200! são tratados do mesmo modo.

A seguir damos algums exemplos do Maple.

Exemplos no MAPLEV.3

Exemplo 1.- Se queremos conhecer a raiz quadrada de 3, com uma precisão de 1000 dígitos temos

```
> >evalf(sqrt(3),1000);
```

$$\sqrt{(3)} := 1.732050807568877293527446341505872366942805\backslash$$
$$2538103806280558069794519330169088000370811461 8\backslash$$
$$6757248575675626141415406703029969945094998952 4\backslash$$
$$7881165551209437364852809323190230558206797482 0\backslash$$
$$1010846749232650153123432669033228866506722546 6\backslash$$
$$8921837971227047131660367861588019049986537379 8\backslash$$
$$5938946765034750657605075661834812960610094760 2\backslash$$
$$1871903250831458295239598329977898245082887144 6\backslash$$
$$3832917347224163984587855397667958063818353666 1\backslash$$
$$1084317378089437831610208830552490167002352071 1\backslash$$
$$1442886959909563657970871684980728994932964842 8\backslash$$
$$3020786408603988738697537582317317831395992983 0\backslash$$
$$0783870287705391336956331210370726401924910676 8\backslash$$
$$2311992883756411414220167427521023729942708310 5\backslash$$
$$9898459475987664288897796147837958390228854852 9\backslash$$
$$0357603385280806438197234466105968972287286526 4\backslash$$
$$1538226646984200211954841552784411812865345070 3\backslash$$
$$5191650016689294415480846071277143999762926834 6\backslash$$
$$2957743836189511012714863874697654598245178855 0\backslash$$
$$9753790138806649619119622229571105552429237231 9\backslash$$
$$2197738262561631468842032853716682938649611917 0\backslash$$
$$49738836395495938$$

Exemplo 2.- Encontrando o fatorial de 700.

```
> factorial(700);
```

$$
\begin{aligned}
&242204012475027217986787509381235221859098338572920 7\backslash\\
&\quad 29945067966492993816021564742044451905166648481\backslash\\
&\quad 92493214566714970498423275250938748173438383937\backslash\\
&\quad 57631459228450828499972271274140160311057830558\backslash\\
&\quad 46363633712407933244782073928110103711266538753\backslash\\
&\quad 71807902575779192731082629169047504052350550600\backslash\\
&\quad 84012219492892375635136296622020023178109619818\backslash\\
&\quad 04617990689745042054891261087058908805650391358\backslash\\
&\quad 45622110376932887829609001950741309997990359707\backslash\\
&\quad 11436279339094292032866260496375825461427727555\backslash\\
&\quad 71000300775290614147063957439002498851491426444\backslash\\
&\quad 98650064588732269519418995459703339103515885592\backslash\\
&\quad 32940829569276986080222002899661283439316300 28\backslash\\
&\quad 78920338265474960347351631476526277225717115468\backslash\\
&\quad 67168628141847287411871479363495016531974574556\backslash\\
&\quad 60413134506049122044947052623384682088864790673\backslash\\
&\quad 30956929238421561178801427495490591414836230322\backslash\\
&\quad 62002468164413019348460802549986473252706061045\backslash\\
&\quad 12088058712293349862185399243309054299576381718\backslash\\
&\quad 80624723819523260464261432989407063616375367209\backslash\\
&\quad 12327516123783482738407578735677175328792425183\backslash\\
&\quad 37119540602943609411629349009566043720836737401\backslash\\
&\quad 09088239297503122461253124564268729671705374773\backslash\\
&\quad 45064433149245581195604799014787362095569251615\backslash\\
&\quad 17737110399754730551854066328420014728657896286\backslash\\
&\quad 93652378708020647632715713644131877343275100726\backslash\\
&\quad 31080569582516938112809572432024601571117786174\backslash\\
&\quad 72683761623869704457588005158037495665069625778\backslash\\
&\quad 93089809572579471070163923823152811557961912028\backslash\\
&\quad 73786892389343351985086659339172571439752777075\backslash\\
&\quad 90597511989345068701735940169672561864713107115\backslash\\
&\quad 01674736899269011608263376217234668896984086251\backslash\\
&\quad 72643840000000000000000000000000000000000000000\backslash\\
&\quad 00000000000000000000000000000000000000000000000\backslash\\
&\quad 00000000000000000000000000000000000000000000000\backslash\\
&\quad 00000000000000000000000000000000000000000
\end{aligned}
$$

Exemplo 3.- O Maple expandindo potências de polinômios

```
%O maple expande pot\^encias de \pols.

> h:=x^4+3*x^3+3*x+4;
```

$$h := x^4 + 3\,x^3 + 3\,x + 4$$

```
> h1:=expand(h^3);
```

$$h1 := x^{12} + 9\,x^{11} + 27\,x^{10} + 36\,x^9 + 66\,x^8 + 153\,x^7 + 135\,x^6 + 153\,x^5 + 264\,x^4 + 171\,x^3 + 108\,x^2 + 144\,x + 64$$

Exemplo 4.- O Maple ordena os polinômios usando o seguinte comando

```
> sort(h1);
```

$$x^{12} + 9\,x^{11} + 27\,x^{10} + 36\,x^9 + 66\,x^8 + 153\,x^7 + 135\,x^6 + 153\,x^5 + 264\,x^4 + 171\,x^3 + 108\,x^2 + 144\,x + 64$$

Exemplo 5.- O Maple cria um polinômio aleatório de qualquer grau e com os coeficientes dentro de um intervalo qualquer.

```
> f:=randpoly(x,degree=18,coeffs=rand(12378..18765));
```

$$f := 14004\,x^{18} + 16201\,x^{16} + 13850\,x^{11} + 16688\,x^8 + 15133\,x^3 + 12946$$

Exemplo 6.- Expandindo e ordenando o produto de dois polinômios.

```
> sort(h2);
```

$$18594\,x^{27} + 180351\,x^{26} + 619083\,x^{25} + 1020519\,x^{24} + 1708526\,x^{23} + 3835155\,x^{22} +$$
$$4991329\,x^{21} + 5564619\,x^{20} + 8623878\,x^{19} + 10013490\,x^{18} + 8991552\,x^{17} +$$
$$10011492\,x^{16} + 10452894\,x^{15} + 8761077\,x^{14} + 7347623\,x^{13} + 5803641\,x^{12} +$$
$$4625192\,x^{11} + 3296352\,x^{10} + 1728952\,x^{9} + 1996956\,x^{8} + 1762020\,x^{7} + 1996956\,x^{6} +$$
$$3445728\,x^{5} + 2231892\,x^{4} + 1409616\,x^{3} + 1879488\,x^{2} + 835328\,x$$

Exemplo 7.- O Maple achando o Máximo Divisor Comum de dois polinômios , usando módulo 127.

```
> Gcd(h1,h2)mod 127;
```

$$x^{12}+9\,x^{11}+27\,x^{10}+36\,x^{9}+66\,x^{8}+26\,x^{7}+8\,x^{6}+26\,x^{5}+10\,x^{4}+44\,x^{3}+108\,x^{2}+17\,x+64$$

Exemplo 8.- O Maple diz quando um polinômio é primitivo módulo 127 ou não .

```
> Primitive(h1)mod 127;
```

$$false$$

```
>Exemplos de C\'alculo;
```

Exemplo 9.-Integrando

```
> int(exp(-x)/(1+x^(3/2)), x=0 .. infinity);
```

Quando o Maple não pode achar uma solução devolve o comando.

$$\int_0^{\infty} \frac{e^{-x}}{1+x^{3/2}} dx$$

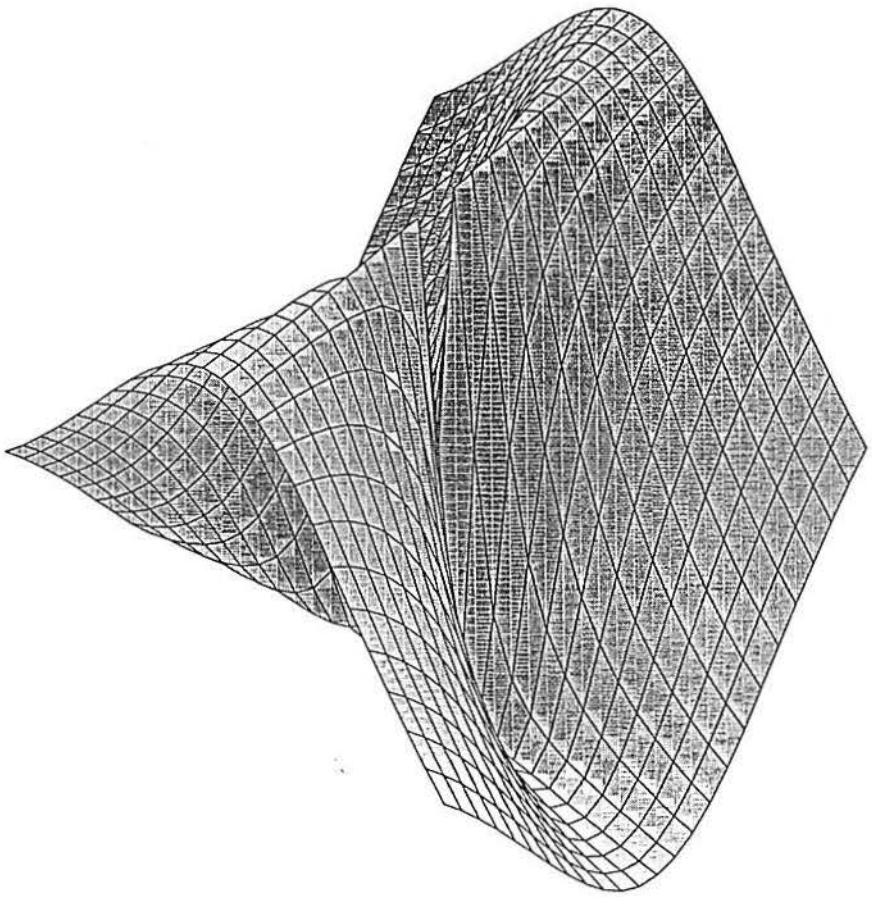
Mas ainda pode-se achar uma solução numérica aproximada dessa integral.

Exemplo 10.- O Maple fazendo gráfico;

```
> grafo:=(r,s) -> (r^2-s^2)/(r^2+s^2);
```

$$grafo := (r,s) \mapsto \frac{r^2 - s^2}{r^2 + s^2}$$

```
> plot3d(grafo,-1..1,-1..1);
```



1.4 Limitações

O que tem-se dito dos sistemas de Computação Algébrica está longe de dizer que estes sistemas oferecem oportunidades ilimitadas e que são um remédio universal para a solução de problemas matemáticos. Esta é uma impressão muito otimista. Os sistemas de computação Algébrica têm tendência a usar muito espaço de memória e tempo computacional. O preço que se paga para ter uma aritmética exata é exponencial.

Exemplo: Para acharmos o MDC de dois polinômios com coeficientes racionais, usando o algoritmo de Euclides, obtém-se coeficientes com 30 digitos, (devido a que usa a divisão longa).

Existem muitas áreas onde o sistema de computação Algébrica não é de muita ajuda, como exemplo temos: Equações diferenciais, integração indefinida e definida com funções de Bessel, integração de contorno, integração de superfícies, álgebras não comutativas.

Para que o uso de um sistema de Computação Algébrica escolhido, seja adequado, precisamos ter familiaridade com as características básicas do Sistema, tais como estrutura e construção de dados, o que faz mais fácil programar eficientemente e adivinhar alguns dos perigos latentes da Computação Algébrica. Espera-se que em breve algumas de estas dificultades sejam superadas.

2 PRELIMINARES

Neste capítulo damos algumas definições e propriedades importantes da Álgebra. A maioria dos resultados são bem conhecidos e podem ser encontrados em qualquer livro texto de Álgebra, (ver [25]). Achamos conveniente colocarmos os resultados aqui, por que eles, em conjunto, representam, com boa aproximação , os preliminares de Álgebra necessários ao estudo computacional em corpos finitos.

Algumas provas são efetuadas quando entendemos que o resultado é importante ou a idéia da prova é interessante.

2.1 Sistemas Algébricos

Nesta seção estudaremos alguns dos sistemas algébricos mais comuns como são Anéis, Domínios e Corpos. Admitimos que o leitor esteja familiarizado com a definição de grupo.

Definição 2.1.1. *Um conjunto R não vazio dotado de duas operações (+, ·), forma um sistema algébrico chamado* **anel**, *se as seguintes propriedades são satisfeitas:*

1. *(R,+) é um grupo abeliano;*
2. *a operação (·) é associativa;*
3. *Para todo a,b,c em R se verifica :*

$$i) \quad a.(b+c) = a.b + a.c$$
$$ii) \quad (a+b).c = a.c + b.c$$

Nota.-

1. R é um *anel comutativo*, se a operação $(\cdot)$, é comutativa.
2. Um Subanel R', de um anel R é um subconjunto não vazio de R, que com as operações de $+$ e $\cdot$ de R, forma um Anel.

Exemplo 2.1.1. *O conjunto dos inteiros módulo m ($\mathbb{Z}_m$) formam um anel.*

Exemplo 2.1.2. *Seja R um Anel comutativo, consideremos*

$$R[x] = \{a_n x^n + ... + a_1 + a_0 : a_n \neq 0, a_i \in R\}$$

Pode-se provar que $R[x]$ junto com as operações bem conhecidas de adição e multiplicação entre polinômios é um Anel comutativo.

Definição 2.1.2. *Seja $a(x) = a_n x^n + ... + a_1 x + a_0 \in R[x]$, então definimos e denotamos o grau de $a(x)$ da seguinte maneira*

$$\partial(a(x)) = \begin{cases} n, & se\ a_n \neq 0 \\ -\infty, & se\ a(x) = 0 \end{cases}$$

Proposição 2.1.1. *Em $R[x]$ verificam-se as seguintes propriedades,*

i) $\partial(a(x).b(x)) \leq \partial(a(x)) + \partial(b(x))$

ii) $\partial(a(x) + b(x)) \leq max\{\partial(a(x)), \partial(b(x))\}$

Prova. Ver [19].

Definição 2.1.3. *A* característica *de um anel R com unidade é a ordem do elemento 1 no grupo aditivo, isto é, a característica* **m** *é finita, se existe $m \in \mathbb{N}$ tal que $m.1 = 0$ e $n.1 \neq 0$, para todo $1 < n < m$.*

Se a característica de R não é finita, então diz-se que tem característica zero.

Exemplo: $\mathbb{Z}_m$ tem característica m.

Exemplo: $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$ têm característica zero.

Entende-se que um anel finito deve ter característica finita, mas a recíproca não é verdadeira.

Exemplo 2.1.3. *$\mathbb{Z}_p[x]$ é um anel infinito que tem característica finita p.*

Definição 2.1.4.

i) Um elemento $a \neq 0$ em R é dito divisor de zero, *se $a.b = 0$, para algum $b \neq 0$ em R (b também é divisor de zero).*

ii) Um elemento $u \neq 0$ em R, é dito unidade, *se u é invertível, isto é, $u.v = 1$, para algum v em R ($v = u^{-1}$).*

Exemplo 2.1.4.

i) Em $\mathbb{Z}_4 = \{0, 1, 2, 3\}$, 2 é divisor de zero, pois $2 \neq 0$ e $2 \times 2 = 0$

ii) 3 é unidade, pois $3 \times 3 = 1$.

Os conceitos de divisor de zero e unidade, nos conduzem a duas classes importantes de anéis comutativos, uma caracterizada por não ter divisor de zeros e outra pela presença de unidades. Primeiramente veremos aqueles que não tem divisores de zero.

Definição 2.1.5. *Um* Domínio de Integridade *D é um anel comutativo não trivial, que não tem divisores de zero; i.e,*

$$a.b = 0 \Rightarrow a = 0 \qquad ou \qquad b = 0.$$

Definição 2.1.6. *Um elemento $x \in D^*$, um Domínio de Integridade é dito* primo, *se $x = a.b (a, b \in D^*)$ e a ou b é uma unidade, de outra maneira chama-se* composto.

Definição 2.1.7. *Um* Corpo *é um anel comutativo, onde cada um dos elementos não nulos é uma unidade.*

Os Corpos são importantes porque neles podemos executar todas as operações aritméticas.

Exemplos

1) $\mathbb{Z}$ é um Domínio de Integridade, mas não um corpo, pois as únicas unidades são 1 e -1.

2) $\mathbb{Q}, \mathbb{R}, \mathbb{C}$ são corpos.

3) $\mathbb{Z}_p$, onde p é um primo, constitui um corpo.

4) $\mathbb{Z}_n$ é um anel com divisores de zero, se n é composto.

Teorema 2.1.1. *Um Domínio de Integridade finito é um Corpo.*

Prova. Seja D um domínio de integridade com elementos diferentes e não nulos, denotados por

$$d_1, d_2, ..., d_n$$

para algum $x \neq 0$ em D, temos

$$xd_1, xd_2, ..., xd_n.$$

Agora cada $xd_i \neq 0$, pois D é domínio de integridade, e como $d_i \neq d_j \Rightarrow xd_i \neq xd_j$, *se* $i \neq j$. De outro modo, teríamos $d_i = d_j$ (pela lei do cancelamento, logo $xd_1, xd_2, ..., xd_n$ são diferentes e não nulos. Em particular, temos:

$$1 = xd_j \qquad \text{para algum} \qquad d_j$$

então $d_j = x^{-1}$ □

Teorema 2.1.2. *Se um Domínio de Integridade tem característica finita m, então m tem que ser primo.*

Prova. Ver [25].

2.2 Ideais e Anéis Quocientes

Definição 2.2.1. *Dado um Anel R, um conjunto I é dito* **Ideal**, *se:*

i) $a - b \in I$, *para todo a,b em I*

ii) $r.a$ *e* $a.r$ *estão em I, para todo* $a \in I$ *e todo* $r \in R$.

Definição 2.2.2. *Seja a um elemento de R, um anel comutativo, então*

$$(a) = \{ra : r \in R\}$$

é o Ideal gerado pelo elemento, a, chamado Ideal Principal.

Exemplo 2.2.1. *Sejam* $a_1, a_2, ..., a_n$ *em R, R anel comutativo, então*

$$(a_1, ..., a_n) = \left\{ \sum r_i a_i : r_i \in R \right\},$$

é um Ideal.

Definição 2.2.3. *Um Domínio é* **Principal**, *se todo ideal é principal.*

Seja R um anel I um ideal de R, definimos a classe de equivalência de um elemento $r \in R$ módulo o ideal I como

$$r + I = \{r + ai : i \in I, a \in R\}$$

É fácil mostrar que o conjunto das classes de equivalência, definidas desse modo, formam um anel, chamado *Anel Quociente.*

Definição 2.2.4. *Seja R um anel e I um Ideal, então*

$$\frac{R}{I} = \{a + I, a \in R\}$$

é o **anel quociente** *módulo I.*

Exemplo 2.2.2. *Nos inteiros $\mathbb{Z}$ temse que (m) é um Ideal, e,*

$$\frac{\mathbb{Z}}{(m)} = \{a + (m) : a \in \mathbb{Z}\},$$

é um Anel Quociente com as operações de classes de equivalência já definidas (chama-se o Anel Quociente de inteiros módulo m), já definido no exemplo (ver 2.1.1).

A seguir damos um teorema que fornece uma notação conveniente para $R[x]/(m(x))$, isto é a notação de classe, $p(x) + (m(x))$ é substituida por notação polinomial.

Lema 2.2.1. *Seja*

$$m(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_0 \ \in \ R[x] \qquad e \qquad (m(x)) = I,$$

então, em $R[x]/I$, $a(x + I) = a(x) + I$.

Teorema 2.2.1. *Em $R[x]$, seja $I = (m(x))$, onde $m(x) = x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$. Seja $\alpha = x + I$, então*

a) $R[x]/I = R(\alpha)$

b) $m(\alpha) = 0$

c) cada $\beta \in R(\alpha)$ pode ser expresso unicamente da forma, $b_0 + b_1\alpha + ... + b_{n-1}\alpha^{n-1}(b_i \in R)$

Prova.

a) Lembre-se que $R(\alpha)$, o anel gerado por $R \cup \{\alpha\}$, consiste de todos os valores $p(\alpha)$ de todos os polinômios $p(x) \in R[x]$. Assim, temos que

$$\begin{aligned} R(\alpha) &= \{p(\alpha) : p(x) \in R[x]\} \\ &= \{p(x+I) : p(x) \in R[x]\} \\ &= \{p(x)+I : p(x) \in R[x]\} \quad \text{(pelo lema 2.2.1)} \\ &= R[x]/I. \end{aligned}$$

b)

$$\begin{aligned} m(\alpha) &= m(x+I) \\ &= m(x)+I \quad \text{(lema 2.2.1)} \\ &= 0+I, \quad \text{pois } m(x) \in I \\ &= 0 \end{aligned}$$

c) Seja $\beta = p(\alpha) \in R[\alpha]$. Para algum $p(x)+(m(x)) \in R[x]/(m(x))$, se prova que

$$p(x)+(m(x)) = r_{m(x)}(p(x))+(m(x))$$

(isto é, cada classe $p(x)+(m(x))$ tem um único representante de grau menor que grau de $m(x)$, denotada por $r_{m(x)}(p(x))$). Logo pelo lema 2.2.1, $p(x+I) = r(x+I)$ (ie. $p(\alpha) = r(\alpha)$). Assim cada $\beta \in R[x]$ pode ser escrito na forma:

$$b_0 + b_1\alpha + ... + b_{n-1}\alpha^{n-1}.$$

Para provar a unicidade, supondo que β tem duas representações

$$\beta = p(x) \qquad e \qquad \beta = q(x), \text{ com } p \neq q,$$

então

$$\begin{aligned} a_0 + a_1\alpha + ... + a_{n-1}\alpha^{n-1} &= b_0 + b_1\alpha + ... + b_{n-1}\alpha^{n-1} \\ p(x+I) &= q(x+I) \\ p(x) + I &= q(x) + I \end{aligned}$$

Mas o grau de $p(x)$ e o grau de $q(x)$ são menores que n; logo como $p(x) + I = q(x) + I$, igualando coeficientes tem-se que $a_0 = b_0, a_1 = b_1, ..., a_{n-1} = b_{n-1}$ (Contradição). Por tanto p(x)=q(x).

□

Exemplo 2.2.3. *Seja*

$$\frac{\mathbb{Z}[x]}{(x^3+1)} = \mathbb{Z}(\alpha) = \{a_0 + a_1\alpha + a_2\alpha^2 : a_i \in \mathbb{Z}\}$$

onde $p(\alpha) = \alpha^3 + 1 = 0$. *Se*

$$p(\alpha) = 1 + 3\alpha^2 \qquad e \qquad q(\alpha) = 4 - 6\alpha,$$

então

$$\begin{aligned} p(\alpha) + q(\alpha) &= 5 - 6\alpha + 3\alpha^2 \\ p(\alpha)q(\alpha) &= 4 - 6\alpha + 12\alpha^2 - 18\alpha^3 \\ &= 22 - 6\alpha + 12\alpha^2 \quad (\textit{usando} \quad \alpha^3 = -1) \end{aligned}$$

2.3 Extensão de Corpos

Definição 2.3.1. *Seja E um corpo, um subanel $F \cup E$ é chamado* subcorpo de E, *se F por si mesmo é um corpo; E é chamado um* Corpo Extensão *de F.*

Definição 2.3.2. *Sejam $\alpha \in E \supseteq F$, dois corpos, α é dito algébrico sobre F, se α satisfaz uma equação polinomial em F, isto é $f(\alpha) = 0$ para algum $f(x) \in F[x]$. $F(\alpha) = F \cup \{\alpha\}$ é dito extensão algébrica de F.*

Observe o seguinte diagrama que ilustra a localização da extensão ,

$$\begin{array}{c} E \\ | \\ F(\alpha) \\ | \\ F \end{array}$$

Exemplo 2.3.1.

1. *$\sqrt[3]{2}$ é algébrico sobre $\mathbb{Q}$, pois $\sqrt[3]{2}$ é raiz de $x^3 - 2 \in \mathbb{Q}$.*

2. *e e π não são algébricos sobre $\mathbb{Q}$.*

Definição 2.3.3. *Seja $\alpha \in E$ algébrico sobre F, definimos o* polinômio mônico mínimo *de α sobre F, denotado por $m_\alpha(x)$ como o polinômio mônico de menor grau que tem α como raiz.*

Propriedades:

1. m_α é único

2. m_α é irredutível

3. Para todo $f(x) \in F[x]$, $\quad f(\alpha) = 0 \Leftrightarrow m_\alpha | f(x)$

Teorema 2.3.1. *Seja α algébrico sobre $F \subseteq E$ cujo polinômio mínimo é $m_\alpha(x)$ sobre F. Então*

$$F(\alpha) \cong F[x]/(m_{\alpha(x)})$$

Prova. Aplicando o teorema Fundamental do Isomorfismo, se podemos achar um

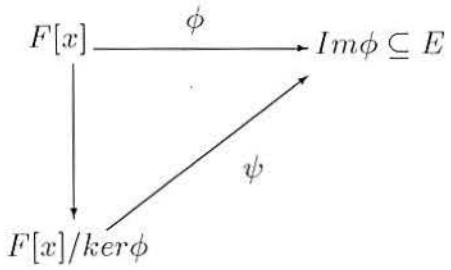


Figura 2.1:

homomorfismo

$\phi : F[x] \to E$ cuja imagem é $F(\alpha)$ e cujo núcleo é $(m_\alpha(x))$, então todo estará feito. O candidato natural para ϕ é:

$$\phi_\alpha(f(x)) = f(\alpha)$$

provaremos que

i) $ker\phi_\alpha = (m_\alpha(x))$

ii) $Im\ \phi_\alpha = F(\alpha)$

De fato,

i) Temos,

$$ker\phi_\alpha = \{f(x) \in F[x] : f(\alpha) = 0\}$$

isto é, o kernel é o ideal de todos os polinômios em $F[x]$ que têm α como uma raiz. Agora qualquer ideal de $F[x]$ é principal, cujo gerador pode ser um polinômio mônico de menor grau. Para o caso de ker_α, este polinômio é por definição , $m_\alpha(x)$ o polinômio mínimo de α sobre F. Assim $ker\phi_\alpha = (m_\alpha(x))$ como queriamos.

ii) Qualquer $f(\alpha) \in Im\phi_\alpha$ deve estar em $F(\alpha)$, por ser fechado, usando que $F(\alpha)$ é um anel contendo $F \cup \{\alpha\}$. Logo $Im\phi_\alpha \subseteq F(\alpha)$. Na outra direção , temos :

$$Im\phi_\alpha \cong F[x]/ker\phi = F[x]/(m_\alpha(x)).$$

Agora $m_\alpha(x)$ é irredutível sobre F o que implica que$F[x]/(m_\alpha(x))$ e por tanto $Im\phi_\alpha$, é um corpo. Assim $Im\phi_\alpha$ é um subcorpo de E que contem cada $a \in F$(pois $\phi_\alpha(a) = a$) e α (pois $\phi_\alpha(x) = \alpha$). Mas então $F(\alpha) \subseteq Im\phi_\alpha$, como $F(\alpha)$, por definição é o menor subcorpo que incluie F, temos que:

$$Im\phi_\alpha = F(\alpha) \qquad \text{como queriamos.}$$

Logo concluimos que:

$$\begin{aligned} \psi : F[x]/(m_\alpha(x)) &\longrightarrow F(\alpha) \\ a(x) + (m(x)) &\longrightarrow a(\alpha) \end{aligned}$$

é um Isomorfismo.

□

Corolario 2.3.1. *Seja $\alpha \in E$ com polinômio mínimo,$m_\alpha(x) = x^n + a_{n-1}x^{n-1} + ... + a_0$. Então todo $\beta \in F(\alpha)$ pode ser unicamente representado na forma:*

$$\beta = b_0 + b_1\alpha + ... + b_{n-1}\alpha^{n-1}, \qquad (b_i \in F).$$

Exemplo 2.3.2.

$$m(x) = x^2 + x + 1 \in \mathbb{Z}$$

$$m(0) = 1, \qquad m(1) = 1,$$

então $m(x)$ *não tem fatores lineares, logo é irredutível onde* $\alpha = x + (x^2 + x + 1)$ *e podemos formar*

$$\mathbb{Z}_2[x]/(x^2 + x + 1), \quad \textit{que é corpo}$$

onde

$$\mathbb{Z}_2[x]/(x^2 + x + 1) = \{a + bx/a, b \in \mathbb{Z}_2\}$$

$$\mathbb{Z}_2(\alpha) = \{0, 1, \alpha, 1 + \alpha\},$$

onde $\alpha^2 + \alpha + 1 = 0$. *As tabelas de adição e multiplicação para* $\mathbb{Z}(\alpha)$ *são*

$+$	0	1	α	$1+\alpha$
0	0	1	α	$1+\alpha$
1	1	0	$1+\alpha$	α
α	α	$1+\alpha$	0	1
$1+\alpha$	$1+\alpha$	α	1	0

$\cdot$	1	α	$1+\alpha$
1	1	α	$1+\alpha$
α	α	$1+\alpha$	1
$1+\alpha$	$1+\alpha$	1	α

Observa-se que $\alpha^{-1} = 1 + \alpha$ *em* $\mathbb{Z}_2[\alpha]$. *Assim temos construido um corpo com 4 elementos,* $|\mathbb{Z}(\alpha)| = 4$.

Exemplo 2.3.3. $x^3 - 2$ *é um polinômio irredutível sobre* $\mathbb{Q}$. *O Anel Quociente:*

$$\frac{\mathbb{Q}}{(x^3 - 2)},$$

é um corpo extensão de $\mathbb{Q}$, *onde* $x^3 - 2$ *tem uma raíz* $\alpha = x + (x^3 + 2)$.

$$\mathbb{Q}(\alpha) = \{a_0 + a_1\alpha + a_2\alpha^2 : \alpha_i \in \mathbb{Q}\},$$

onde $\alpha^3 - 2 = 0$.

Da divisão de $x^3 - 2$ *pelo fator linear* $x - \alpha$, *obtemos uma fatoração explícita de* $x^3 - 2$ *(usando* $\alpha^3 = 2$*):*

$$x^3 - 2 = (x - \alpha)(x^2 + \alpha x + \alpha^2)$$

Então em $\mathbb{Q}(\alpha)$, $x^3 - 2$ *é redutível.*

Teorema 2.3.2. *Seja* $f(x) \in F[x]$ *um polinômio de grau* n *maior ou igual a 1. Então* F *pode ser estendido para um corpo* E *no qual* $f(x)$ *tem* n *raízes.*

Prova. Por indução sobre n.

i) $n = 1$, então $E = F$.

ii) Seja $n \geq 1$ e supomos que a afirmação é verdadeira para polinômios de grau $k \leq n$. Agora seja f de grau $n + 1$; existe uma extensão E de F que contém uma raiz α de f (ver [25]). Portanto

$$f(x) = (x - \alpha)g(x), \qquad \text{em} \quad E, \qquad \text{com} \quad \partial(g) = n$$

pela hipótese de indução existe uma extensão K de E onde g tem n raízes de f e logo f tem $n + 1$ raízes em K. □

Definição 2.3.4. *Um corpo de raízes sobre* F, *de um polinômio* $f(x) \in F[x]$ *é um corpo* E *que satisfaz*

1. *Todas as raízes de* $f(x)$ *estão em* E
2. E *é a menor extensão de* F *que satisfaz (1).*

Teorema 2.3.3. *Qualquer polinômio $f(x) \in F[x]$ tem um corpo de raízes sobre F.*

Prova.-ver [19]

2.4 Teorema Chinês do Resto

Nesta seção estudaremos este teorema que é bastante conhecido. Existem dois casos no qual usaremos este teorema, o anel dos inteiros e polinômios. Estes dois casos podem agrupar-se num conjunto mais abstrato como são os Domínios Euclidianos. (ver [11]).

Teorema 2.4.1. *Sejam $m_1, m_2, ..., m_r$ inteiros positivos que são primos relativos em pares; isto é para $0 \leq j, k \leq r$,*

$$gcd(m_j, m_k) = 1, \qquad j \neq k,$$

Sejam $m = m_1.m_2...m_r$ e $a, u_1, u_2, ..., u_r$ inteiros. Então existe um único inteiro u que satisfaz as condições: $a \leq u < a + m$, e

$$\begin{aligned} u &\equiv u_1 \mod m_1 \\ &\vdots \\ u &\equiv u_r \mod m_r \end{aligned}$$

Prova. Por definição, se

$$u \equiv v \mod m_j, \qquad 1 \leq j \leq r$$

então

$$u - v = k_j m_j, \qquad \text{para todo } j$$

como os m_j são primos entre si, então temos:

$$u - v = km_1.m_2.....m_r = km.$$

Portanto existe no máximo uma solução.

Para completar a prova, mostremos que existe ao menos uma solução, e usamos a forma construtiva por adequar-se a métodos computacionais.Pretendemos achar um método prático para converter $(u_1, u_2, ..., u_r)$ até U, segundo Garner (ver [22]) diz que pode levar-se a cabo usando só

$$\begin{aligned} \binom{r}{2} &= \frac{r\,!}{(r-2)\,!2\,!} \quad \text{constantes } c_{ij}, \\ \text{para } 1 &\leq i < j \leq r, \text{ onde} \\ c_{ij}m_i &\equiv 1 \mod m_j \end{aligned} \tag{2.1}$$

As c_{ij} obtém-se usando o Algoritmo Estendido de Euclides que nos permite achar a e b tais que

$$am_i + bm_j = 1$$

e podemos fazer $a = c_{ij}$. Toda vez que se acham as c_{ij} satisfazendo 2.1, fazemos,

$$\begin{aligned} v_1 &\leftarrow u_1 \mod m_1 \\ v_2 &\leftarrow (u_2 - v_1)c_{12} \mod m_2 \\ v_3 &\leftarrow ((u_3 - v_1)c_{13} - v_2) \mod m_3 \\ &\vdots \\ v_r &\leftarrow (...(u_r - v_1)c_{1r} - v_2)c_{2r} - v_{r-1})c_{r-1,r} \mod m_r \end{aligned}$$

Então

$$u = v_r m_{r-1}...m_2m_1 + ... + v_3m_2m_1 + v_2m_1 + v_1$$

é um número que satisfaz as condições:

$$\{0 \leq u < m, \qquad u \equiv u_j \mod m_j, \qquad 1 \leq j \leq r\}$$

□

Um teorema Chinês dos Restos mais geral pode ser enunciado:

Teorema 2.4.2. *Sejam $u_1, u_2, ..., u_r$ polinômios sobre um corpo F, com*

$$MDC(u_j(x), u_k(x)) = 1, \quad j \neq k$$

Para quaisquer $w_1(x), w_2(x), ..., w_r(x)$ polinômios em $F[x]$, existe um único polinômio $v(x) \in F[x]$ tal que

$$\partial(v(x)) < \partial(u_1(x)) + ... + \partial(u_r(x))$$

e

$$\begin{aligned} v(x) &\equiv w_1(x)(\mod u_1(x)) \\ &\vdots \\ v(x) &\equiv w_r(x)(\mod u_r(x)) \end{aligned}$$

Prova.- Ver [22].

Exemplo.- Dados

$$m_1 = 3; \quad m_2 = 5; \quad m_3 = 7$$

e

$$u_1 = 1; \quad u_2 = 6; \quad u_3 = 5,$$

achar um número u que satisfaz as condições do teorema 2.4.1, usando o algoritmo extendido de Euclides (ver [22]pp.325-327), obtemos

$$c_{1,2} = 2; \quad c_{1,3} = 1; \quad c_{2,3} = -2,$$

usando o algoritmo de Garner temos que,

$$v_1 = 1; \quad v_2 = 0; \quad v_3 = 6,$$

e

$$u = v_3 m_2 m_1 + v_2 m_1 + v_1 = 91.$$

Assim temos que 91 cumple com todas as condições do teorema Chinês dos Restos.

3 CORPOS FINITOS

Neste capítulo trataremos de caracterizar corpos finitos. Usaremos resultados desenvolvidos no capítulo anterior, para mostrar como são e que ordem têm essas estruturas algébricas, que são o ambiente de trabalho para os algoritmos que apresentamos no capítulo seguinte.

3.1 Introdução

Já vimos que $\mathbb{Z}_p$, o anel dos inteiros módulo p é um corpo com p elementos, se p é primo. Este fato é uma conseqüência do teorema 2.1.1, que diz que todo domínio de integridade finito é um corpo.

Esses corpos são muito importantes em Computção Algébrica, por várias razões . A principal é que a aritmética em $\mathbb{Z}_p$ é simples e principalmente eficiente, pois o tamanho dos números é limitado por p. Esse fato permite que vários problemas sobre os inteiros sejam *mapeadas* módulo p e resolvidos aí nesse domínio. Tais procedimentos são chamados de Métodos Modulares. Exemplos de problemas resolvidos modularmente são fatoração de polinômios inteiros, Bases de Grobner e até o isolamento de raízes reais. Na busca da caracterização dos corpos finitos, veremos sua estrutura e como construi-los, basicamente como extensões algébricas de $\mathbb{Z}_p$. Nesse sentido as operações entre seus elementos já estão definidas como no capítulo anterior.

Vamos mostrar que os corpos finitos têm cardinalidade uma potência de um número primo. Além disso, os corpos de mesma ordem são isomorfos, quer dizer que existe apenas um corpo de cada ordem possível. Além disso, mostraremos que os corpos finitos são cíclicos, isto é todos os seus elementos são gerados a partir de um elemento.

Como os corpos finitos são essencialmente únicos, uma notação especial é usada, $GF(q)$ **denota o "Corpo de Galois de q elementos", com** $q = p^n$

3.1.1 Estrutura Cíclica dos Corpos Finitos

Mostraremos agora o resultado principal sobre a estrutura dos corpos finitos que seu grupo mltiplicativo é cíclico.

Lema 3.1.1. *Sejam a e b elementos de grupo abeliano com ordem m e n, respectivamente. Se $MDC(m,n) = 1$ então , ordem de $a \cdot b = m \cdot n$*

Prova.-Ver [19].

Teorema 3.1.1. *Seja G um grupo abeliano finito e seja m a ordem de algum elemento de ordem maximal de G. Então a ordem de qualquer elemento de G divide m.*

Prova.-Ver [19].

Teorema 3.1.2. *Seja $GF(q)$ um corpo finito. Então $GF(q)^* = GF(q) - \{0\}$, o grupo multiplicativo de $GF(q)$, é cíclico.*

Prova. Seja $|GF(q)| = q$ o número de elementos de $GF(q)$, e seja $\alpha \in GF(q)^*$ com ordem maximal m.
Queremos mostrar que $m = q-1$ ou $m \leq q-1$ *ou* $m \geq q-1$).
Pelo teorema de Lagrange, a ordem de qualquer elemento de $GF(q)^*$ divide a ordem de $GF(q)^*$, então $m \leq q-1$.
Agora para mostrar $m \geq q-1$, examinamos as raízes de $x^m - 1$ *em* $GF(q)^*$. Para qualquer $b \in GF(q)^*$, $o(b) = n$, logo $n|m$ pelo teorema 3.1.1, assim que $m = kn$. Então

$$b^m = (b^n)^k = 1$$

logo $b^m - 1 = 0$, isto é cada elemento de $GF(q)^*$ é uma raíz de $x^m - 1$. Mas $x^m - 1$ tem no máximo m raízes distintas, logo $m \geq q - 1$. □

Qualquer gerador do grupo cíclico $GF(q)^*$ é um chamado de elemento primitivo de $GF(q)$. Assim temos provado que qualquer corpo finito tem um elemento primitivo. A questão de achar um elemento primitivo, é importante para a construção do corpo finito e, por conseguinte, para a eficiência da aritmética nesse corpo. Questões probabilísticas são importantes aqui. Dado um elemento $\alpha \in GF(p)$, qual a probabilidade de ser primitivo. O seguinte resultado nos fornece de uma ferramenta nessa direção .

Teorema 3.1.3. *Seja $|GF(q)| = q$. Então $GF(q)$ tem $\phi(q-1)$ elementos primitivos, onde ϕ é a função de Euler.*

Prova.- Ver [25].
Por exemplo num corpo de 9 elementos, $GF(3^2)$, existem $\phi(8) = 4$ geradores.

3.2 Corpos Finitos como Extensões Algébricas

Aplicaremos a teoria de extensão de corpos à teoria de corpos finitos, com o objetivo de descrever a estrutura das extensões finitas. A propridade cíclica tem influência dominante sobre a estrutura dos corpos finitos, como poderemos observar.

Lema 3.2.1. *Seja $GF(q)$ um corpo finito $|GF(q)| = q$. Então para todo $\beta \in GF(q)^*$ é raiz de $x^{q-1} - 1$.*

Prova. $|GF(q)^*| = q - 1$
ora para todo $\beta \in GF(q)^*$, $o(\beta) \mid q - 1$ (pelo teorema de Lagrange), isto é,

$$k\ o(\beta) = q - 1 \qquad \text{e logo} \quad \beta^{q-1} = \beta^{k\ o(\beta)} = (\beta^{o(\beta)})^k = 1.$$

□

Como corolário óbvio, vale o seguinte teorema,

Teorema 3.2.1. *Seja F corpo finito e $GF(q)$ uma extensão finita de F. Então para todo $\beta \in GF(q)$ é* algébrico simples *sobre F.*

Teorema 3.2.2. *Seja F um corpo finito e $GF(q)$ uma extensão finita de F, então*

$$GF(q) \cong F[x]/(m_\alpha(x))$$

onde $m_\alpha(x)$ é o polinômio de um elemento primitivo α de $GF(q)$.

Prova. Do teorema 3.2.1, $GF(q)$ é $F(\alpha)$ a extensão simples de F, então

$$F(\alpha) \cong \frac{F[x]}{(m_\alpha(x))}, \text{ pelo teorema 2.3.1.}$$

Pelo corolário 2.3.1 do teorema 2.3.1, cada $\beta \in GF(q)$, extensão finita de F, pode ser escrito de forma única como

$$\beta = b_0 + b_1\alpha + ... + b_{n-1}\alpha^{n-1}, \quad b_i \in F$$

e

$$n = \partial(m_\alpha(x)).$$

□

Isso nos garanta o seguinte resultado:

Corolario 3.2.1. *Seja F corpo finito com $|F| = p$ e $GF(q)$ uma extensão finita. Então $|GF(q)| = p^n$, para algum inteiro $n > 0$*

Exemplo: Supondo $p = 2$, $n = 4$, então analisamos o Corpo $GF(2^4)$.
Sejam $\mathbb{Z}_2 = \{0, 1\}$ e o polinômio $p(x) = x^4 + x + 1$, verifica-se que $p(x)$ é irredutível.

Agora $p(x) = x^4 + x + 1$ tem uma raiz α em uma extensão de $\mathbb{Z}_2$,

$$\mathbb{Z}_2[x]/(x^4 + x + 1),$$

e também $x^4 + x + 1$ é o polinômio mínimo de α sobre $\mathbb{Z}_2$. Onde cada elemento de $GF(2^4)$ pode-se representar de uma única maneira:

$$\beta = a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3, \quad a_i \in \mathbb{Z}_2 \quad \text{é} \quad GF(2^4).$$

Assim $\mathbb{Z}_2(\alpha) = \mathbb{Z}/(x^2 + x + 1)$ é um corpo Extensão de $\mathbb{Z}_2$ com 16 elementos. Observemos uma tabela 3.2 dos elementos de $\mathbb{Z}_2(\alpha)$ junto com seus polinômios mínimos sobre $\mathbb{Z}_2$. Sabemos que $\alpha^4 + \alpha + 1 = 0$

β	α_0	α_1	α_2	α_3	Polinômio mínimo sobre $\mathbb{Z}_2$
0	0	0	0	0	x
1	1	0	0	0	$x+1$
α	0	1	0	0	x^4+x+1
α^2	0	0	1	0	x^4+x+1
α^3	0	0	0	1	$x^4+x^3+x^2+x+1$
$1+\alpha=\alpha^4$	1	1	0	0	x^4+x+1
$\alpha+\alpha^2=\alpha^5$	0	1	1	0	x^2+x+1
$\alpha^2+\alpha^3=\alpha^6$	0	0	1	1	$x^4+x^3+x^2+x+1$
$1+\alpha+\alpha^3=\alpha^7$	1	1	0	1	$x^4+x^3+x^2+x+1$
$\alpha+\alpha^2+\alpha^4=\alpha^8$	1	0	1	1	x^4+x^3+1
$\alpha+\alpha^3=\alpha^9$	0	1	0	1	x^4+x+1
$\alpha^2+\alpha^4=\alpha^{10}$	1	1	1	0	$x^4+x^3+x^2+x+1$
$\alpha^3+\alpha^2+\alpha=\alpha^{11}$	0	1	1	1	x^2+x+1
α^{12}	1	1	1	1	x^4+x^3+1
α^{13}	1	0	1	1	$x^4+x^3+x^2+x+1$
α^{14}	1	0	0	1	x^4+x^3+1
$1=\alpha^{15}$	1	0	0	0	$x+1$

Tabela: **Uma Representação de** $GF(2^4)$

Queremos verificar que α^5 tem polinômio mínimo x^2+x+1 sobre $\mathbb{Z}_2$, então notemos que x^2+x+1 é irredutível sobre $\mathbb{Z}_2$, pois não tem raízes em $\mathbb{Z}_2$, também

$$\begin{aligned}(\alpha^5)^2+\alpha^5+1 &= \alpha^{10}+\alpha^5+1\\ &= (\alpha^2+\alpha+1)+(\alpha^2+\alpha)+1(\text{Da tabela.})\\ &= 0,\end{aligned}$$

se reduz para zero, usando aritmética módulo 2, e assim e mostramos que $x^2 + x + 1$ é polinômio mínimo de α^5. Da mesma maneira se verifica para os outros polinômios. Na tabela observamos que as potências α^i formam $GF(2^4)$; devido a que o polinômio irredutível x^4+x+1 tem um elemento primitivo de $GF(2^4)$ como raiz, tal polinômio é chamado Primitivo. Em geral um polinômio irredutível de grau n sobre $GF(q)$ é dito Primitivo se tem um elemento primitivo de $GF(q^n)$ por raiz.

Podemos verificar que nem todo polinômio irredutível é primitivo, por exemplo
$x^4 + x^3 + x^2 + x + 1$ é irredutível sobre $\mathbb{Z}_2$. Assim cada elemento de,

$$GF(2^4) \cong \mathbb{Z}_2/(x^4 + x^3 + x^2 + x + 1)$$

pode representar-se como,

$$a_0 + a_1\beta + a_2\beta^2 + a_3\beta^3, \quad (a_i \in \mathbb{Z}_2).$$

onde $\beta^4 + \beta^3 + \beta^2 + \beta + 1 = 0$, mas β não é primitivo.
A ordem de β é 5, pois

$$\begin{aligned} \beta^4 + \beta^3 + \beta^2 + \beta + 1 &= 0 \\ \beta^5 + \beta^4 + \beta^3 + \beta^2 + \beta &= 0 \\ \beta^4 + \beta^3 + \beta^2 + \beta + 1 + 1 &= \beta^5 \\ 0 + 1 &= \beta^5 \end{aligned}$$

3.3 Existência e Unicidade de Corpos Finitos

Já vimos no capítulo anterior que se um domínio de Integridade tem característica finita esta tem que ser prima. Por tanto um corpo finito tem que ter necessariamente característica um número primo.

É fácil ver que um corpo $GF(q)$ de característica p contém um subcorpo de p elementos que podemos identifica-lo como $\mathbb{Z}_p$. Com essa identificação aplicada ao teorema 3.2.2, podemos caracterizarmos corpos finitos:

Teorema 3.3.1. *Qualquer corpo finito $GF(q)$ tem p^n elementos para algum primo p e algum $n \in \mathbb{N}$. Onde p é característica de $GF(q)$ e n é o grau do polinômio mínimo de um elemento primitivo α de $GF(q)$ sobre $\mathbb{Z}_p$.*

Ainda existe a questão da unicidade e existência. Por exemplo $\mathbb{Z}_2/(x^2 + x + 1)$ e $\mathbb{Z}_2/(x^4 + x + 1)$ são isomorfos?. Para cada p e cada n existe um corpo finito com p^n elementos?.
Para respondermos tais questões , necessitamos de algums resultados auxiliares. Agora mostraremos a unicidade dos corpos finitos.

Lema 3.3.1. *Sejam $a_1(x), a_2(x), ..., a_l(x)$ fatores distintos irredutíveis de $f(x) \in F[x]$, então*

$$\prod_{l=1}^{L} a_l(x) \mid f(x)$$

Proposição 3.3.1. *Seja $GF(q^n)$ com elementos $\alpha_0 = 0, \alpha_1, \alpha_2, ..., \alpha_Q$. Então sobre $GF(q^n)$, $x^Q - 1$ tem a seguinte fatoração irredutível:*

$$x^Q - 1 = \prod_{i=1}^{Q}(x - \alpha_i)$$

Proposição 3.3.2. *Sejam $m_1(x), ..., m_l(x)$ todos os polinômios mínimos distintos dos elementos x de $GF^*(q^n)$ sobre $GF(q)$.*
Então sobre $GF(q)$, $x^Q - 1$ tem a seguinte fatoração:

$$x^Q - 1 = \prod_{l=1}^{L} m_l(x)$$

Prova. Ver [25].

Corolario 3.3.1. *Se $\alpha \in GF(q^n)$ tem polinômio mínimo $m_\alpha(x)$ sobre $GF(q)$, então $m_\alpha(x)$ tem raízes distintas em $GF(q^n)$*

Exemplo 3.3.1. *Se*
$Q = 2^4 - 1$, *então* $x^{15} - 1 = x^Q - 1$, *queremos a fatoração irredutível,*
i) Sobre $GF(2^4)$, *temos*

$$x^{15} - 1 = \prod_{i=0}^{14}(x - \alpha^i),$$

onde α *é um elemento primitivo de* $GF(2^4)$, *isto é,* α *é uma raiz de* $x^4 + x + 1$.
ii) Sobre $GF(2)$

$$x^{15} - 1 = (x+1)(x^2+x+1)(x^4+x+1)$$
$$(x^4+x^3+1)(x^4+x^3+x^2+x+1)$$

O próximo resultado garante a unicidade de $GF(q)$.

Teorema 3.3.2. *(Unicidade de $GF(p^n)$) Quaisquer dois corpos com p^n elementos são isomorfos.*

Prova. Se F é um corpo com p^n elementos, então tem característica p, logo F é uma extensão de $\mathbb{Z}_p$. Pela proposição anterior, fazendo $Q = p^n - 1$, $x^{p^n-1} - 1$ tem a seguinte fatoração irredutível sobre $\mathbb{Z}_p$;

$$x^{p^n-1} - 1 = \prod_{l=1}^{L} m_l(x) \tag{3.1}$$

Onde os $m_l(x)$ são polinômios mínimos distintos dos elementos de F^*.

Escolhendo α um elemento primitivo de F, então

$$F = \mathbb{Z}_p(\alpha) \cong \mathbb{Z}_p[x]/(m_\alpha(x))$$

Suponhamos agora que G é um outro corpo com p^n elementos e por exatamente as mesmas razões que as de acima $x^{p^n-1}-1$, tem a seguinte fatoração irredutível sobre $\mathbb{Z}_p$

$$x^{p^n-1}-1=\prod_{l=1}^{\bar{L}}\bar{m}_l(x) \tag{3.2}$$

Onde os $\bar{m}_l(x)$ são os polinômios mínimos sobre $\mathbb{Z}_p$ de G^*. Mas 3.1 e 3.2 conducem a duas fatorações de $x^{p^n-1}-1$. Pela unicidade da fatoração, a familia de $m_l(x)$ deve coincidir com a familia de $\bar{m}_l(x)$. Em particular $m_l(x)=m_\beta(x)$ para algum $\beta \in G$. Logo pelo teorema anterior

$$\mathbb{Z}_p(\beta) \cong \frac{\mathbb{Z}_p[x]}{m_\alpha(x)}.$$

Como $\dfrac{\mathbb{Z}_p}{m_\alpha(x)} \cong F$ tem p^n elementos, assim como o subcorpo $\mathbb{Z}_p(\beta)$ de G, por tanto

$$G \cong \mathbb{Z}_p(\beta) \cong F$$

□

Resta-nos mostrar a existência de corpos finitos, isto é,

Teorema 3.3.3. *Para cada primo p e inteiro $n > 0$, existe um corpo com p^n elementos.*

Prova. Consideremos o polinômio

$$f(x)=x^{p^n}-x \quad \text{em} \quad \mathbb{Z}_p[x].$$

Então $f(x)$ tem um corpo de raízes 2.3.2 : Uma extensão $GF(q)$ de $\mathbb{Z}_p$, que contém todas as raízes de $f(x)$ é tal que nenhum subcorpo, próprio de $GF(q)$ tenha todas as raízes de $f(x)$. Agora mostremos que o corpo de raízes de $f(x)$ tem exatamente p^n

elementos. Observamos que $f(x)$ tem p^n raízes não necessariamente distintas. Como

$$f'(x) = p^n x^{p^n-1} - 1 = -1 \neq 0,$$

pelo criterio de multiplicidade de raízes, temos que as p^n raízes são distintas. Provaremos que as raízes de $f(x) = x^{p^n} - x$ forman um corpo. De fato,
Se r_1, r_2 são raízes de $f(x)$, então,

$$\begin{aligned} f(r_1 + r_2) = (r_1 + r_2)^{p^n} - (r_1 + r_2) & \\ &= r_1^{p^n} + r_2^{p^n} - (r_1 + r_2), \\ &= r_1 + r_2 - (r_1 + r_2), \\ &= 0 \end{aligned}$$

Assim temos provado que as raízes de $f(x)$ são fechadas com respeito da adição, e se

$$\begin{aligned} f(r_1, r_2) &= (r_1 r_2)^{p^n} - r_1 r_2 \\ &= r_1^{p^n} r_2^{p^n} - r_1 r_2 \\ &= r_1 r_2 - r_1 r_2 = 0, \end{aligned}$$

é fechado com respeito ao produto. É fácil verificar as outras propriedades de corpo. Portanto, temos construído um corpo com p^n elementos, o corpo de raízes de $x^{p^n} - x$ sobre $\mathbb{Z}_p$. □

3.4 Aritmética em Corpos finitos

Teremos aqui alguns fatos importantes sobre a aritmética de corpos finitos. Um primeiro fato bem conhecido, é o chamado de Pequeno Teorema de Fermat

que junto com suas conseqüencias, é uma das razões mais importantes da eficiência da aritmética em Corpos finitos.

Lema 3.4.1. *Todo elemento* $a \in GF(p)$ *satisfaz* $a^p = a$.

Prova.- Ver [19].

Lema 3.4.2. *Em um domínio de característica* p, *temos*

$$(a+b)^p = a^p + b^p$$

Prova.

$$(a+b)^p = \sum_{k=0}^{p} \binom{p}{k} a^k b^{p-k}$$

o k-ésimo termo $0 \leq k \leq p$:

$$\binom{p}{k} a^k b^k = \frac{p(p-1)...(p-(k-1)}{k!} a^k b^{p-k}$$

ora como

$$\binom{p}{k} \quad \text{é inteiro}$$

$$k\,! \mid p(p-1)...(p-k+1) \quad \text{mas} \quad MDC(k\,!, p) = 1$$

assim $k\,! \mid p(p-1)...(p-k+1)$. e cada k-ésimo termo entre 0 e p é do tipo $p.m = 0$ pois o domínio tem característica p. □

Corolario 3.4.1. $(a+b)^{p^n} = a^{p^n} + b^{p^n}$ *em um domínio de característica* p.

Na seção anterior caracterizamos corpos finitos como sendo anéis quocientes de $\mathbb{Z}_p[x]/(m(x))$ com $m(x)$ polinômio irredutível em $\mathbb{Z}_p[x]$.
Ainda resta a tarefa de encontrarmos um polinômio irredutível de grau n e em $\mathbb{Z}_p$. O resultado seguinte nos dá um critério de irredutibilidade.

Lema 3.4.3. *Sejam* $l_1, ..., l_k$, *todos os divisores primos de* n *e denotemos* $\frac{n}{l_i} = m_i$. *Um polinômio* $g(x) \in \mathbb{Z}_p[x]$ *de grau* n *é irredutível em* $\mathbb{Z}_p[x]$ *se e somente se,*

i) $g(x) \mid x^{p^n} - x$

ii) $MDC(g(x), x^{p^{m_i}} - x) = 1, \qquad 1 \geq i \geq k$

Prova.- Ver [19]

O próximo resultado nos mostra a quantidade de polinômios irredutíveis de um grau dado.

Lema 3.4.4. *Denotemos por $m(n)$ o número dos diferentes polinômios mônicos em $\mathbb{Z}_p[x]$ de grau n que são irredutíveis. Então*

$$\frac{p^n - p^{n/2\log n}}{n} \leq m(n) \leq \frac{p^n}{n}$$

$$\frac{1}{2n} \leq \frac{m(n)}{p^n} \cong \frac{1}{n}$$

onde p^n é o número de todos os polinômios mônicos de grau n.

Prova.- Ver [27].

OPERAÇÕES

- Os elementos em

$$\mathbb{Z}_p[x] \mod (m(x)) = \{p(x) \in \mathbb{Z}_p[x] : \partial(p(x)) < \partial(m(x))\}$$

- A soma e resta são definidas como em $\mathbb{Z}_p[x]$, e o produto e o elemento inverso se definem da seguinte maneira,

$$p(x) * q(x) = r_{m(x)}(p(x)q(x))$$

$$\frac{1}{p(x)} = s(x) \mod m(x)$$

onde $MDC(p(x), m(x)) = 1$ e $p(x)s(x) = 1$

- O elemento zero e a unidade são,

$$(m(x)) = \{m(x)p(x) : p(x) \in \mathbb{Z}_p[x]\}$$

e

$$1 + (m(x)).$$

A seguir apresentamos dois algoritmos para determinarmos polinômios irredutíveis de grau n em $\mathbb{Z}_p[x]$, com isso concluindo a tarefa de construção de $GF(p^n)$.

3.4.1 Algoritmo de Rabin para Polinômios Irredutíveis

Algoritmo 3.1.

Repetir:

(1) Gerar um polinômio mônico aleatório, $g(x)$, de grau n sobre $GF(p)$.

(2) Se $g(x) \mid (x^q - x) = 1$, então a condição (i) ocorre.

(3) Se $MDC(g(x), (x^{p_i^n} - x))$ para todo $n_i = n/k_i$, onde os k_i são todos os divisores primos de n, então a condição (2) ocorre.

Até que (1) e (2) (condições de 3.4.3 ocorram.

A prova deste algoritmo é direto de 3.4.3.

3.4.2 Algoritmo de Calmet para Polinômios Irredutíveis

Em [14] se prova que um método melhor resulta da substituição do passo (1) e (2), e a construção da matriz Q de Berlekamp,(ver capítulo seguinte)

Algoritmo 3.2.

Repetir:

(1) Gerar um polinômio mônico aleatório, $g(x)$ de grau n sobre $GF(p)$

(2) Se $g(x)$ é livre de quadrados, então o passo (1) ocorre.

(3) Se o passo (1) ocorre, construir a matriz Q segundo 4.5. Se o número de vetores linearmente independentes é 1, então o passo (2) ocorre.

Até que o passo (2) ocorre.

O tempo médio computacional para o método de Rabin é $o(n^4(\log\ p)^3)$, e o de Calmet é levemente melhor, $o(n^4(\log\ p)^2 + n^3(\log\ p)^3)$.(Ver [12])

4 FATORAÇÃO DE POLINÔMIOS EM UMA VARIÁVEL

Apresentamos e desenvolvemos algoritmos para a fatoração de polinômios em corpos finitos.

4.1 Introdução

A fatoração polinomial é uma operação importante na manipulação algébrica, é importante não somente como maneira de achar raízes ou fatores, mas também por ser usada como subalgoritmo em outros processos tais como a integração simbólica, simplificação , solução de equações polinomiais, etc. Naturalmente desejamos ter um método rápido e eficiente (e assim nos dirigimos a estudar a eficiência dos algoritmos para fatorar).

Todos os problemas de fatoração polinomial são reduzidos ao problema de fatoração de polinômio mônicos com coeficientes inteiros a uma variável, mesmo problemas de fatoração de polinômios a várias variáveis em extensões algébricas.

O método de Kronecker (ver [22]) foi o primeiro método usado para fatorar polinômios univariáveis sobre os inteiros. Mas o algoritmo é ineficiente e se mostra que o custo cresce exponencialmente com o grau do polinômio a ser fatorado.

Esta ineficiência fez que se desenvolvessem métodos homeomórficos. Estes métodos reduzem o problema para o caso de uma variável, com o polinômio reduzido módulo um primo p. O polinômio resultante é então fatorado sobre o corpo finito $\mathbb{Z}_p = GF(p)$. Alguns destes fatores serão usados para determinar os fatores sobre $\mathbb{Z}$.

Vemos portanto que a fatoração de polinômios em corpos finitos é importante não só por si mesmo, mas também como um subalgoritmo para vários métodos homomórficos.

Varios métodos são considerados neste trabalho, entre os quais, o método de Berlekamp, o de Cantor-Zassenhaus, e o de Rabin (ver [15], [22]) . Nosso problema principal é achar um fator sobre $\mathbb{Z}_p$. Muito do trabalho nesta área foi feito por Berlekamp [5], ele desenvolveu o primeiro algoritmo de Fatoração completa que trabalha na ordem de $o(n^3p)$ passos, para fatorar um polinômio de grau n sobre $\mathbb{Z}$. Uma das desvantagem deste método foi o termo p usado na análise do tempo. Isto restringiu o método para corpos relativamente pequenos. Depois o mesmo Berlekamp [6] refinou seu método de tal maneira que o problema de fatorar se reduz a encontrar raízes de um polinômio num corpo finito. Ele mostra que o problema pode ser resolvido num tempo proporcional a $p^{1/4}\log p^{3/2}$.

Neste capítulo estudaremos métodos para a decomposição de polinômios numa variável em corpos finitos. Inicialmente veremos como é possível obter um algoritmo eficiente para uma decomposição livre de quadrados, a seguir varios métodos probabilísticos para a fatoração serão discutidos.

Dado $U(x) \in GF(q)[x]$ onde F é um corpo finito, é possível obter únicos polinômios distintos $u_1(x), u_2(x), ..., u_n(x)$, irredutíveis com coeficientes em $GF(q)$, de tal modo que

$$U(x) = u_1^{s_1}(x).u_2^{s_2}....u_s^{s_s}(x)$$

onde $s_1, s_2, .., s_s$ são inteiros maiores do que zero. A garantia da existência e unicidade dos u_i é dado pelo fato de $GF(q)[x]$ ser um Domínio Euclidiano portanto de Fatoração única.

Neste capítulo trataremos justamente de como obter os u_i . Entretanto a maioria dos métodos requerem que a potência de u_i seja 1, isto é, que o polinômio seja livre de quadrados. Por isso precisamos conhecer tal decomposição.

4.2 Decomposição livre de quadrados

Seja um polinômio $p(x)$ de $GF(q)[x]$ onde R é um anel de característica zero. É posível que $p(x)$ tenha fatores múltiplos, quer dizer, que exista um polinômio q tal que q^2 divida $p(x)$ (talvez numa potência maior que 2 divida $p(x)$, ainda neste caso é verdade que q^2 divide a $p(x)$).Obviamente podemos encontrar todos os fatores múltiplos fazendo uma fatoração completa de $p(x)$, mas existe uma forma mais simples que descreveremos. É suficiente considerar $p(x)$ mônico. Suponhamos que $p(x)$ seja fatorado num produto de fatores lineares:

$$p(x) = \prod_{i=1}^{n}(x - a_i)^{n_i},$$

onde os a_i podem ser quantidades algébricas sobre R (fazemos esta fatoração só para a prova). A derivada de $p(x)$ é:

$$p'(x) = \sum_{i=1}^{n}(n_i(x - a_i)^{n_i-1} \prod_{\substack{j=1 \\ i\neq j}}^{n}(x - a_j)^{n_j}).$$

Supondo que $p'(x) \neq 0$, é óbvio que para cada i, $(x - a_i)^{n_i-1}$ divide $p(x)$ e $p'(x)$. Por outro lado cada polinômio que divide $p(x)$ é um produto dos $(x-a_i)$ para uma potência menor ou igual que n_i. Por tanto o MDC (máximo divisor comum) determinado é o produto dos $(x - a_i)$, para uma potência $n_i - 1$ ou n_i. Mas esta potência não pode ser n_i, pois $(x-a_i)^{n_i}$ divide todos os termos de $p'(x)$ exceto um, e portanto não pode dividir $p'(x)$. Assim que temos provado o seguinte resultado:

$$d(x) = MDC(p(x), p'(x)) = \prod_{i=1}^{n}(x - a_i)^{n_i-1},$$

Assim

$$q(x) = \frac{p(x)}{MDC(p(x), p'(x))} = \prod_{i=1}^{n}(x - a_i)$$

Então

$$MDC(q(x), MDC(p(x), p'(x))) = \prod_{\substack{i=1 \\ n_i>1}}^{n} (x - a_i),$$

pelo que deduzimos

$$h(x) = \frac{q(x)}{MDC(q(x), MDC(p(x), p'(x)))} = \prod_{i=1 n_i=1}^{n} (x - a_i).$$

O lado direito desta equação é o produto de todos os fatores não-múltiplos de $p(x)$ e mostramos como calcula-lo usando só operações de derivação, divisão exata e MDC. Estas operações não sairam de $GF(q)[x]$ o que implica que este produto pode ser calculado em $GF(q)[x]$ por um método eficiente e rápido. Em resumo, calculamos como um resultado intermediário o $MDC(p(x), p'(x))$, que tem os mesmos fatores múltiplos de $p(x)$, mas com a potência reduzida em 1, o mesmo procedimento que se aplicó à $p(x)$ agora é aplicado ao $MDC(p(x), p'(x))$, então estamos calculando todos os fatores de $MDC(p(x), p'(x))$ de multiplicidade um, é dizer os fatores de $p(x)$ de multiplicidade 2. E similarmente para os fatores de multiplicidade 3,...etc. isto é, por meio de cálculos simples em $GF(q)[x]$ podemos fatorar da forma $\prod p(x)i_i$, onde $p_i(x)$ é o produto de todos os fatores de $p(x)$ de multiplicidade i. Nesta decomposição de $p(x)$, cada $p_i(x)$ não tem fatores múltiplos, e os $p_i(x)$ são relativamente primos entre si. Esta decomposição é conhecida com o nome de *Livre de quadrados*,ver[30].

Se $U'(x) = 0$, então o coeficiente a_k de x^k, somente é diferente de zero se k é múltiplo de q. Isso quer dizer que $p(x)$ pode ser escrito como $U(x^q)$, ou seja

$$p(x) = U(x^q) = (U(x))^q.$$

Nesse caso a decomposição pode ser feita em $U(x)$ tomando as q-ésimas potências de $U(q)$.

Exemplo 4.1. *Seja o seguinte polinômio em* $\mathbb{Z}_{13}$

$$\mathrm{p}(\,x\,) := x^9 + 3\,x^8 + 6\,x^7 + 9\,x^6 + 12\,x^5 + 4\,x^4 + 5\,x^3 + 2\,x^2 + 11\,x + 4$$

E seja $p1(x)$ *a derivada de* $p(x)$

$$p1(x) := 9x^8 + 24x^7 + 42x^6 + 54x^5 + 60x^4 + 16x^3 + 15x^2 + 4x + 11$$

Usamos o método de Decomposição livre de quadrados para eliminar o fatores múltiples de $p(x)$. *Se o* MDC *é igual a 1, então* $p(x)$ *não têm fatores múltiples, de outro modo* $p(x)$ *têm fatores repetidos.*
Agora achando o MDC *de* $p(x)$ *e* $p1(x)$ *temos,*

$$d(x) := x^4 + 11x^3 + 10x^2 + 4x + 4$$

Assim fazendo a divisão de $p(x)$ *com* $d(x)$ *obtemos o polinômio livre de quadrados,*

$$q(x) := x^5 + 5x^4 + 6x^3 + 6x^2 + 5x + 1$$

Achando o $MDC(q(x), d(x))$, *obtemos o fatores de multiplicidade 3.*

$$q1(x) := x^2 + 12x + 11$$

E dividendo $q(x)$ *entre o* $q1(x)$ *temos os fatores de multiplicidade 1.*

$$h(x) := x^3 + 6x^2 + x + 6$$

Finalmente conhecemos os fatores $(x^3 + 6x^2 + x + 6)$ *de multiplicidade 1* $(x^2 + 12x + 11)$ *de multiplicidade 3 E o polinômio livre de quadrados será o produto de*

$$(x^3 + 6x^2 + x + 6)(x^2 + 12x + 11)$$

Verificamos multiplicando os fatores anteriores usando suas multiplicidades como potências.

$$p(x) = (x^3 + 6x^2 + 6)(x^2 + 12x + 11)^3.$$

$$\mathrm{p}(x) := x^9 + 3x^8 + 6x^7 + 9x^6 + 12x^5 + 4x^4 + 5x^3 + 2x^2 + 11x + 4$$

4.3 O Método de Berlekamp

Seja $U(x) = u_n x^n + ... + u_1 x + u_0$, um polinômio com coeficientes no conjunto $\{0, 1, 2, ..., p-1\}$. Usando aritmética módulo **p**, queremos expressar U(x) num produto de polinômios irredutíveis. Assumimos que U(x) é livre de quadrados, isto é, cada fator tem no máximo potências 1.

Portanto, como estamos num Domínio de Fatoração Única, supomos que

$$U(x) = p_1(x)p_2(x)....p_r(x) \tag{4.1}$$

é o produto de fatores primos entre si.

Para descobrir os $p_j(x)$ a partir de U(x), a idéia de Berlekamp é fazer uso do Teorema Chinês dos Restos, que é válido tanto para polinômios como para inteiros. Se $(s_1, s_2, ..., s_r)$ é uma r-upla de inteiros módulo **p**, o T.C.R. implica que existe um único $V(x)$ tal que,

$$\begin{cases} V(x) \equiv s_1 \bmod p_1(x) \\ V(x) \equiv s_2 \bmod p_2(x) \\ \quad \vdots \\ V(x) \equiv s_r \bmod p_r(x) \end{cases} \tag{4.2}$$

onde,

$\partial(V) < \partial(p_1) + \partial(p_2) + ... + \partial(p_r) = \partial(U)$. O conhecimento deste polinômio dá uma maneira de obter os fatores de U(x). Por exemplo, se $r = 2$ e $s_1 \neq s_2$,

$MDC(U(x), V(x) - s_1)$ é divisível por $p_1(x)$ e não por $p_2(x)$. Já observamos que pode-se obter informação a respeito de dos fatores de U(x), partindo de soluções apropriadas para o sistema 4.2, tratamos agora de obter ditas soluções
Em primeiro lugar, observa-se que V(x) satisfaz a condição :

$$\begin{cases} V(x)^p \equiv s_j^p = s_j \equiv V(x) \mod p_j(x) \\ V(x)^p \equiv V(x) \pmod{U(x)} \end{cases} \tag{4.3}$$

onde, $1 \leq j \leq r$ e $\partial(V) \leq \partial(U)$ Em segundo lugar, temos a identidade polinomial básica:

$$x^p - x \equiv (x-0)(x-1)...(x-(p-1)) \mod p$$

e daí segue que

$$V(x)^p - V(x) = V(x)(V(x)-1)...(V(x)-(p-1)) \tag{4.4}$$

Se V(x) satisfaz 4.3, segue que $U(x)$ divide o lado esquerdo de 4.4, (por definição de módulo); assim cada fator irredutível de $U(x)$ deve dividir um dos p-fatores primos do lado direito da equação em 4.4.

Em outras palavras todas as soluções de 4.3 devem ter a forma de 4.2, ou seja para alguns $s_1, s_2, ..., s_r$ existe exatamente p^r soluções de 4.3 . As solução $V(x)$ para 4.3 são essenciais para a fatoração de $U(x)$. Até pode parecer mais difícil achar todas as soluções para 4.3, que fatorar $U(x)$, mas isto não é verdade, como veremos a seguir.
Seja $\partial(U) = n$, podemos construir a matriz $n \times n$:

$$Q = \begin{pmatrix} q_{0,0} & q_{0,1} & \cdots & q_{0,n-1} \\ \vdots & \vdots & \vdots & \vdots \\ q_{n-1,0} & q_{n-1,1} & \cdots & q_{n-1,n-1} \end{pmatrix}$$

onde,

$$x^{pk} = q_{k,n-1}x^{n-1} + q_{k-1,n-2}x^{n-2} + \ldots + q_{k,1}x + q_{k,0} \pmod{U(x)}, \quad k = 0, ..., n-1.$$

Então

$$V(x) = v_{n-1}x^{n-1} + \cdots + v_1 x + v_0$$

é uma solução para ?? se e somente se,

$$(v_0, v_1, ..., v_{n-1})Q = (v_0, v_1, ..., v_{n-1})$$

A última equação se segue de:

$$V(x) = \sum_j v_j x^j = \sum_j \sum_k v_k q_{k,j} x^j \equiv \sum_k v_k x^{pk} = V(x^p) = V(x)^p \pmod{U(x)}.$$

É importante notar, portanto, que o problema de acharemos soluções $V(x)$ de 4.2 é reduzido a um problema de álgebra linear, de resolução de equações lineares.

O posto da matriz $Q - I$ é o número de fatores irredutíveis.

A seguir apresentamos o algoritmo de Berlekamp.

4.3.1 O Algoritmo de Berlekamp

Para fatorar procede-se da seguinte maneira:

Algoritmo 4.1.

B1.- Constatar que $U(x)$ é livre de quadrados.

B2.- Formar a Matriz Q.

B3.- Triangularizar a matriz $Q - I$, onde I é a identidade $n \times n$, achando seu posto r e seus vetores linearmente independentes: $v^{[1]}, v^{[2]}, ..., v^{[r]}$, tal que

$$v^{[j]}(Q - I) = (0, 0, ..., 0), \qquad 1 \leq j \leq r$$

O primeiro vetor é sempre (1,0,...0) e representa a solução trivial $v^{[1]}(x) = 1$ para ??; r é o número de fatores irredutíveis de $U(x)$. As soluções para ?? são os p^r polinômios correspondentes aos vetores $t_1 v^{[1]}, ..., t_r v^{[r]}$, para todas as escolhas $0 \leq t_1, ..., t_r < p$. Portanto se $r = 1$, $U(X)$ é irredutível e o procedimento acaba.

B4.- Calcular

$$MDC(U(x), v^{[2]}(x) - s) \quad para \quad 0 \leq s < p$$

onde $v^{[2]}(x)$, é o polinômio representado por $v^{[2]}$, o resultado será uma fatoração não trivial de $U(x)$, pois $v^{[2]}(x) - s \neq 0$ e tem o grau menor que o grau de $U(x)$. Sempre e quando $V(x)$ satisfaz ??, prova-se a seguinte equação:

$$U(x) = \prod_{0 \leq s < p} MDC(V(x) - s, U(x))$$

Se quando usaremos $v^{[2]}(x)$ não separa $U(x)$ em r-fatores, os fatores seguintes podem-se obter calculando

$$MDC(v^{[k]}(x) - s, w(x)) \quad para \quad 0 \leq s < p$$

e todos os fatores de $w(x)$ serão encontrados, para $k = 3, 4, ...$ até que os r fatores sejam obtidos. Se escolhemos $s_i \neq s_j$ em 4.2, obtem-se a solução $V(x)$ para ?? que distingue $p_i(x)$ de $p_j(x)$; algum $v^{[k]}(x) - s$ será divisível por $p_i(x)$ e não por $p_j(x)$, assim este procedimento achará todos os fatores.

Exemplo 4.3.1. *Fatorar:*

$$U(x) = x^8 + x^6 + 10x^4 + 10x^3 + 8x^2 + 2x + 8 \pmod{13}$$

usando o algoritmo de Euclides temos:

$$MDC(U(x), U'(x)) = 1$$

Por tanto $U(x)$ *é* livre de quadrados, *não tem fatores repetidos, se não fosse assim usar o método de Decomposição livre de quadrados, portanto o passo B1 é satisfeito. No passo B2 formamos a matriz* Q *de ordem* 8×8. *Podemos obter as filas de* Q *encontrando solução para:*

$$\begin{aligned} x^0 &\mod U(x) \\ x^{13} &\mod U(x) \\ x^{26} &\mod U(x) \\ &\vdots \\ x^{91} &\mod U(x) \end{aligned}$$

Em geral para obter $x^k \mod U(x)$ *temos: Seja*

$$U(x) = x^n + u_{n-1}x^{n-1} + ... + u_1x + u_0$$

Para valores pequenos de k, $x^k \mod U(x)$ *pode ser achada da seguinte forma:*

$$x^k \equiv a_{k,n-1}x^{n-1} + ... + a_{k,1}x + a_{k,0}x \mod U(x)$$

Então

$$\begin{aligned} x^{k+1} &\equiv a_{k,n-1}x^n + ... + a_{k,1}x^2 + a_{k,0}x \\ &= a_{k,n-1}(-u_{n-1}x^{n-1} - ... - u_1x - u_0) + a_{k,n-2}x^{n-1} + ...a_{k,0}x \\ &= (-a_{k,n-1}u_{n-1} + a_{k,n-2}x^{n-1} + ... + a_{k,0} - a_{k,n-1}u_1)x - a_{k,n-1}u_0. \end{aligned}$$

ou seja para $0 \leq j \leq n-1$ *temos a seguinte fórmula :*

$$a_{k+1,j} = a_{k,j-1} - a_{k,n-1}u_j$$

admitindo que:

$$a_{k,-1} = 0 \quad e \quad a_{k+1,0} = -a_{k,n-1}u_0$$

Por exemplo obtemos o coeficiente de x^7 *usando a fórmula anterior*

$$a_{8,1} = a_{7,0} - a_{7,7}u_1 = -2 = 11 \pmod{13}.$$

e assim achamos a matriz Q.

No passo B3, obtemos o posto da matriz $Q - I$, *onde* r *é o número de vetores linearmente independentes:* $v^{[1]}, ..., v^{[r]}$ *tal que,*

$$v^{[2]}A = v^{[2]}A = ... = v^{[2]}A = (0, ..., 0),$$

onde $A = Q - I$ *usando um algoritmo apropriado. (Knuth sugere o algoritmo do espaço nulo) ([22]pp.425). Em nosso exemplo achamos r=3 , então* $U(x)$*deve ter exatamente 3 fatores irredutíveis:*

$$\begin{aligned} v^{[1]} &= (1, 0, ..., 0) \\ v^{[2]} &= (0, 5, 5, 0, 9, 5, 1, 0) \\ v^{[3]} &= (0, 9, 11, 9, 10, 12, 0, 1) \end{aligned}$$

Finalmente no passo B4 , ao calcular $MDC(U(x), v^{[2]}(x) - s)$, *para* $0 \leq s < 13$, *onde* $v^{[2]}(x) = x^6 + 5x^5 + 9x^4 + 5x^2 + 5x$, *e obtemos para:*

$$\begin{aligned} s = 2 \quad &\rightarrow x^3 + 8x^2 + 4x + 12 \\ s = 0 \quad &\rightarrow x^5 + 5x^4 + 9x^3 + 5x + 5 \end{aligned}$$

Portanto com $v^{[2]}$ *obtemos só dois dos tres fatores,(para os outros valores de s, temos que o MDC é a unidade). Então agora usamos o terceiro vetor e fazemos o mesmo procedimento.*

De

$$MDC(v^{[3]} - s, x^5 + 5x^4 + 9x^3 + 5x + 5)$$

obtemos,

$$\begin{aligned} s = 6 \quad &\rightarrow x^4 + 2x^3 + 3x^2 + 4x + 6 \\ s = 8 \quad &\rightarrow x + 3 \end{aligned}$$

Portanto

$$U(x) = (x^4 + 2x^3 + 3x^2 + 4x + 6)(x^3 + 8x^2 + 4x + 12)(x + 3)$$

4.3.2 O custo do Algoritmo

Vamos a estimar o tempo de rodagem do método de Berlekamp, quando um polinômio de grau n é fatorado módulo p. Primeiro assumimos que p é relativamente pequeno ($p < 25$), de tal maneira que as quatro operações aritméticas podem ser feitas módulo p em essencialmente uma certa quantidade de tempo, proporcional a 1.

A computação do passo B1, toma $o(n^2)$ unidades de tempo, o passo B2 toma $o(pn^2)$, para o passo B3, usar o algoritmo para triangulizar matrizes requer

no máximo $o(n^3)$ unidades de tempo. Finalmente no passo B4 observamos que a computação do $MDC(f(x), g(x))$, via algoritmo de Euclides, toma o(grau(f)· grau(g)) unidades de tempo, logo a computação do $MDC(v^{[j]}(x) - s, w(x))$ para j, s fixos e para todos os fatores $w(x)$ de $U(x)$, se encontram no máximo em $o(n^2)$ unidades de tempo. Potanto no passo B4, requer-se de $o(prn^2)$ unidades de tempo como máximo.

O procedimento de Berlekamp para fatorar requer $o(n^3 + prn^2)$ passos, quando p é um primo pequeno.

4.4 O Método de Cantor-Zassenhaus

Em 1981, Cantor e Zassenhaus ([15]) introduz um novo algoritmo *probabilístico* para fatorar polinômios , cujo tempo esperado de rodagem é polinomial em n, o grau de f, e $\log q$, $q = p^n$, isto é q é a cardinalidade do corpo GF(q), tornando-o adequado para valores grandes de q, o que faz deste procedimento mais eficiente que o algoritmo *determinístico*[1] de Berlekamp, cujo tempo de rodagem é proporcional a q, o que não é bom quando q é grande.

Algoritmo 4.2 (CZ).

CZ1.- Verificar que f não tem fatores repetidos, é mônico e $f(0) \neq 0$.

CZ2.- Fatorar f num produto:

$$f(x) = \prod_{i=1}^{n} h_i(x).$$

onde cada $h_i(x)$ contém fatores irredutíveis de grau i.

CZ3.- Fatorar cada $h_i(x)$

[1] chama-se determinístico porque testa todos os posíveis valores de $GF(q)$

Para executar CZ2 temos que usar o algoritmo chamado *Fatoração de grau diferente* e no passo CZ3 usaremos o algoritmo *Fatoração de grau uniforme*, ambos serão estudados a seguir.

4.5 Fatoração de grau diferente

O algoritmo de Fatoração de grau diferente(DDF) se baseia no seguinte lema

Lema 4.5.1. *O polinômio $x^{q^m} - x$ é o produto de todos os polinômios mônicos, irredutíveis e distintos em $GF(q)[x]$ com grau dividindo m.*

Este algoritmo revela o número e os graus dos fatores irredutíveis de f.

Algoritmo 4.3 (DDF).

DDF1.- *Seja* $i \leftarrow 1$; $r_0(x) \leftarrow x$.

DDF2.- *Seja* $r_i(x) \leftarrow (r_{i-1})^q \mod f(x)$

DDF3.-

Seja $h_i(x) \leftarrow MDC(f(x), r_i(x) - x)$

Se $h_i \neq 1$, *faça* $f(x) \leftarrow \frac{f(x)}{h_i(x)}$

Se $f(x) = 1$, *acaba.*

DDF4.-

Seja $i \leftarrow i + 1$

Se $2i \leq$ *grau (f(x)), vai para o passo 2,*

de outro modo, faça $h_{grau(f)} \leftarrow f(x)$, *logo acaba.*

4.5.1 Custo do Algoritmo

O maior custo do algoritmo está no passo 2, em geral para elevar $r_i(x)$ a uma potência grande, se usa o método binario [5] (o método binário para q^m com q primo e aritmética modular é ótimo).

O número de multiplicações módulo $f(x)$ é de ordem, $(\log\ q)^2$.

Baseados no método de Berlekamp, se recomenda fazer o seguinte procedimento: supondo que

$$r_{i-1}(x) = b_{n-1}x^{n-1} + ... + b_1x + b_0 \ \in \ GF(q)[x]$$

então, se

$$r_i(x) = (r_{i-1}(x))^q = (r_{i-1}(x)^q)$$

Portanto se calculamos:

$$Q_i(x) = x^{ip} \mod f(x), \qquad i = 0, 1, ..., n-1$$

e armazenamos Q_i como sendo a i-ésima fila da matriz $Q_{n\times n}$, temos:

$$r_i(x) = r_{i-1}(x).Q \tag{4.5}$$

Ao formar a matriz Q, 4.5 dá uma forma rápida para calcular $r_i(x)$ a partir de r_{i-1}. Segundo [22]pp.426, podemos completar a matriz Q, usando o seguinte método: Seja

$$x^q \mod f(x) = c_{n-1}x^{n-1} + ... + c_1x + c_0$$

Então

$$\begin{aligned} x^{q+1} \mod f(x) &= c_{n-1}x^n + ... + c_1x^2 + c_0x = (c_{n-2} - a_{n-1}c_{n-1})x^{n-1} + \quad (4.6) \\ &+ (c_{n-3} - a_{n-2}c_{n-1})x^{n-2} + ... + (c_0 - a_1c_{n-1})x + a_0c_{n-1} \quad (4.7) \end{aligned}$$

O custo para completar cada linha é: np multiplicações, e para toda a matriz Q é : n^2q operações em $GF(q)$.

Em forma geral se pode determinar $x^k \mod f(x)$ com o uso da seguinte fórmula de recurrência:

$$c_{k+1,j} = c_{k,j-1} - a_j c_{k,n-1}$$

onde

$$c_{k,-1} = 0, \qquad c_{k+1,0} = -c_{k,n-1}a_0$$

Quando se acha Q, se faz a seguinte reformulação do algoritmo DDF.

4.5.2 O algoritmo NDDF.

Algoritmo 4.4 (NDDF).

NDDF1.-

Faça $i \leftarrow 1$;

$r(x) \leftarrow x^q \mod f(x)$

NDDF2.-

Faça $h_i \leftarrow MDC(f(x), r(x) - x)$,

NDDF3.-

Se $h_i(x) = f(x)$, *então acaba;*

de outro modo $f(x) \leftarrow \dfrac{f(x)}{h_i(x)}$, *e* $i \leftarrow i + 1$

NDDF4.-

Se $2i > grau(f(x))$, *então* $h_{grau(f)} \leftarrow f(x)$ *e acaba;*

de outro modo, $r(x) \leftarrow r(x).Q$ *e volta para o passo 2.*

Usaremos o mesmo exemplo dado para Berlekamp.

Exemplo 4.2. *Seja o polinômio*

$$f(x) = x^8 + x^6 + 10x^4 + 10x^3 + 8x^2 + 2x + 8 \pmod{13}$$

Aplicando o algoritmo NDDF temos

```
> nddf(pol,13);
```

$$i = 1$$

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 7 & 11 & 10 & 12 & 5 & 11 \\ 3 & 6 & 4 & 3 & 0 & 4 & 7 & 2 \\ 4 & 3 & 6 & 5 & 1 & 6 & 2 & 3 \\ 2 & 11 & 8 & 8 & 3 & 1 & 3 & 11 \\ 6 & 11 & 8 & 6 & 2 & 7 & 10 & 9 \\ 5 & 11 & 7 & 10 & 0 & 11 & 7 & 12 \\ 3 & 3 & 12 & 5 & 0 & 11 & 9 & 12 \end{bmatrix}$$

$$\begin{bmatrix} 1 & x & x^2 & x^3 & x^4 & x^5 & x^6 & x^7 \end{bmatrix}$$

$$11 + 3\,x + 2\,x^2 + 6\,x^3 + 10\,x^4 + 12\,x^5 + 9\,x^6$$

$$i = 2$$

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 7 & 11 & 10 & 12 & 5 & 11 \\ 3 & 6 & 4 & 3 & 0 & 4 & 7 & 2 \\ 4 & 3 & 6 & 5 & 1 & 6 & 2 & 3 \\ 2 & 11 & 8 & 8 & 3 & 1 & 3 & 11 \\ 6 & 11 & 8 & 6 & 2 & 7 & 10 & 9 \\ 5 & 11 & 7 & 10 & 0 & 11 & 7 & 12 \\ 3 & 3 & 12 & 5 & 0 & 11 & 9 & 12 \end{bmatrix}$$

$$[\,1 \quad x \quad x^2 \quad x^3 \quad x^4 \quad x^5 \quad x^6 \quad x^7\,]$$

$$2 + 10\,x + 5\,x^2 + 12\,x^3 + 12\,x^4 + 7\,x^6 + 4\,x^7$$

$$i = 3$$

$$[\,x + 3, 1, x^3 + 8\,x^2 + 4\,x + 12, x^4 + 2\,x^3 + 3\,x^2 + 4\,x + 6\,]$$

f foi decomposto num produto de fatores, onde todos os subfatores têm o mesmo grau. Agora analisamos como decompor estes s fatores, $x + 3$ deve ter fatores de grau 1

$x^3 + 8x^2 + 4x + 12$ deve ter fatores de grau 3

$x^4 + 2x^3 + 3x^2 + 4x + 6$ deve ter fatores de grau 4. Neste caso, cada um dos fatores é irredutível e portanto não será necessário usar o seguinte algoritmo

4.6 Fatoração de grau Uniforme

Seja

$$f(x) = \prod_{i=1}^{n} h_i,$$

onde cada $h_i(x)$ contem só polinômios irredutíveis de grau i, mônicos, e supondo que

$$h(x) = h_i(x) = x^l + d_{l-1}x^{l-1} + ... + d_1x + d_0,$$

temos a tarefa de achar os l/i fatores de grau i. A fatoração seria trivial se $l = i$

Definição 4.6.1. .- *Um polinômio t(x) (aleatório)* $\in GF(q)[x]$, *se chama polinômio* Separador *para* $h(x)$, *se* $0 <$ *grau(MDC(t,h))* $< l$.
Agora o problema fica reduzido a achar polinômios separadores para $h(x)$.

Daremos dois separadores para o polinômio $h(x)$.

4.6.1 O Algoritmo Separador de Cantor Zassenhaus

Algoritmo 4.5 (CZSEP).

1.- *Escolher um polinômio em forma aleatória,* $t(x)$, *em* $GF(q)[x]$ *de grau menor que* $2i - 1$.

2.- *Fazer* $t(x) \leftarrow (t(x))^{q^i-1/2} \mod h(x)$

3.- *Calcular*

$$g(x) = MDC(h(x)) = MDC(h(x), t(x) - 1)$$

Em Knuth [33] prova-se que o polinômio g(x) tem a probabilidade de quase 1/2 para que seja não trivial. Se $h(x) \neq g(x) \neq 1$, então substituimos $h(x)$ por

$h(x)/g(x)$. Se o grau de $g(x)$ ou $h(x)$ é maior que i, aplicamos de novo o algoritmo. O processo termina quando achamos todos os $m = l/i$ fatores irredutíveis de grau i.

Exemplo 4.3. *Seja o polinômio* $h(x) := x^4 + 3x^3 + 3x + 4$ *em* $\mathbb{Z}_5[x]$.

```
> sepcz(h,2);
{--> enter sepcz, args = x^4+3*x^3+3*x+4, 2
```

$$h := x^4 + 3\,x^3 + 3\,x + 4$$

$$l := 4$$

$$m := 2$$

$$c0 := 1$$

$$lt := 2$$

$$t := 4\,x^2 + 3\,x + 2$$

$$t1 := 3\,x + x^3 + 4\,x^4 + 3\,x^5 + 4\,x^6 + 3\,x^8 + 2\,x^9 + x^{15} + x^{18} + 3\,x^{16} + 1 + 2\,x^{21} + 4\,x^{20} + 4\,x^{19} + x^{24} + 4\,x^{23}$$

$$t := 3\,x^3 + x^2 + 3\,x + 2$$

$$g := x^2 + 1$$

$< --$ *como resultado temos dois fatores do mesmo grau* x^2+1, x^2+3x+4

Prova. (Probabilidade de 1/2)

Se $V(x)$ é uma solução para **??**, sabemos que $V(x)$ divide

$$V(x)^q - V(x) = V(x)(V(x)^{(q-1/2)} + 1)(V(x)^{(q-1/2)} - 1).$$

Isto sugere que calculemos:

$$MDC(U(x), V(x)^{(q-1/2)} - 1) \tag{4.8}$$

Com uma pouca de sorte 4.8 será um fator não trivial de $U(x)$. De fato podemos determinar exatamente quanto de sorte está envolvida; considerando 4.2, seja

$$V(x) \equiv s_j \mod q_j(x), \qquad \text{para} 1 \leq j < r;$$

então $q_j(x)$ divide $V(x)^{(q-1)/2} - 1$, se e somente se,

$$s_j^{(q-1)/2} \equiv 1 \mod q.$$

Sabemos que exatamente $(q-1)/2$ dos inteiros s no intervalo $0 \leq s < q$ satisfaz

$$s^{(q-1)/2} \equiv 1 \mod q,$$

logo a metade dos $q_j(x)$ aparecerá no MDC 4.8.

Precisamente, se $V(x)$ é uma solução aleatória de 4.8, onde todos os q^r soluções são parecidas, a probabilidade de que o MDC seja o $U(x)$ é exatamente

$$((q+1)/(2p))^r,$$

e a probabilidade de que seja igual a 1 é, $((q-1)/2p)^r$.

A probabilidade de que um fator não trivial será obtida é portanto

$$1 - (\frac{q-1}{2p})^r - (\frac{q+1}{2p})^r = \tag{4.9}$$

$$= 1 - \frac{1}{2^{r-1}}\left(1 + \binom{r}{2}q^{-2} + \binom{r}{4}q^{-4} + \cdots\right) \geq \frac{4}{9}, \tag{4.10}$$

para todo $r \geq 2$ e $q \geq 3$. □

4.6.2 Custo do Algoritmo

A complexidade do passo 2 deste algoritmo é dominante. Note que elevar um polinômio $t(x)$ a potência $q^i \mod h(x)$ é muito mais fácil do que elevar o mesmo polinômio a potência $(q^i - 1)/2 \mod h(x)$ o que se necessita para este procedimento. Observe-se que

$$\frac{q^i - 1}{2} = \frac{q-1}{2}(1 + q + ... + q^{i-1}).$$

Portanto para elevar $t(x)$ a potência $q^i - 1/2 \mod h(x)$ necessitamos $i - 1$ multiplicações polinomiais módulo $f(x)$, a um custo de n^2 multiplicações em cada $GF(q)$. Elevar $t(x)$ as potências $q, ..., q^{i-1} \mod h(x)$ pode ser executado em l^2 multiplicações em $GF(q)$, usando a matriz Q gerada por $h(x)$.

O custo total para CZSEP é portanto

$$(2(i-1) + o(\log q))l^2$$

e o tempo esperado para achar todos os fatores de $h_i(x)$ é

$$o(l^2 m(i + \log q)) = o(l^3 + l^2 m \log q)$$

operações em $GF(q)$, onde $m = l/i$ é o número de fatores irredutíveis de $h_i(x)$

4.6.3 Algoritmo Separador de Ben-Or

E um processo muito similar ao de Cantor-Zassenhaus; também é probabilístico. Dados

$$h(x) = h_i(x) \qquad e \qquad g(x) \in GF(q)[x], \text{polinômio aleatório}$$

O operador **Mc Eliece** é definido por:

$$tr(g) = g + g^q + ... + g^{q^{l-1}}$$

Ben-Or mostra que $tr(g)$ é um polinômio separador para $h(x)$ com probabilidade $1 - \frac{1}{q^{m-1}}$, onde $m = l/i$ é o número de fatores irredutí veis de $h_i(x)$. Ele também mostra que com uma probabilidade de $1 - \frac{m(m-1)}{2q}$, $tr(g)$ separará todos os fatores de $h(x)$. Assim, $h(x)$ pode ser fatorado mediante o algoritmo seguinte

Algoritmo 4.6.

(1) Escolher um polinômio aleatório $g(x)$
em $GF(q)[x]$ *de grau* $\leq l$

(2) Fazer $s(x) \leftarrow tr(g) \mod h(x)$.

(3) Se grau(s)=0 então vai para passo (1).

(4) Se grau(h)=i retorne $h(x)$.

(5) Fazer $s(x) \leftarrow s(x) \mod h(x)$.

(6) Se grau(s)=0 então vai para passo (1).

(7) Para um t aleatório em $GF(q)$,
fazer $g(x) \leftarrow ((s(x) + t)^{(q-1)/2} - 1)$

(8) *Fazer* $g(x) \leftarrow MDC(g(x), h(x))$

(9) *Se (grau(g) = 0 ou grau(g) = grau(h))*
então vai para passo (7)

(10) *Vai para o passo (4) com* $h \leftarrow g$ *e* $h \leftarrow h/g$

A desvantagem deste algoritmo radica em que o candidato separador, $tr(g)$, que é fácilmente obtido, tem que ser elevado varias vezes á potência $(q-1)/2$, que é uma operação custosa. O custo do Algoritmo é similar ao de Cantor-Zassenhaus.

5 ENCONTRANDO RAíZES DE POLINÔMIOS EM CORPOS FINITOS

Desenvolvemos algoritmos para o cálculo de raízes polinomiais em corpos finitos.

5.1 Introdução

Naturalmente que raízes de polinômios podem ser obtidas como fatores lineares, portanto aplicando os algoritmos do capítulo anterior. Entretanto, é de se esperar que o seu cálculo seja mais simples que fatoração . De fato, vários algoritmos de fatoração são baseados no cálculo de raízes polinomiais em extensões finitas.

Nesta capítulo estudamos alguns algoritmos modulares para achar raízes de polinômios em $GF(q)$, apresentados por Moenck [26] e Rabin [28], ambos são métodos que melhoram o método de fatoração de Berlekamp [6], um método baseado no cálculo de raízes polinomiais da unidade, isto é geradores de corpos finitos , no final apresentaremos um algoritmo de Rabin.

Definição 5.1.1 (raiz n-ésima primitiva da unidade). *Seja R um Anel, $\omega \in R$ e chamado raiz n-ésima primitiva da unidade se*

1. $\omega \neq 1$
2. $\omega^n = 1$
3. $\omega^k \neq 1, \quad se \quad k < n)$

Lema 5.1.1. *Para todo $a \in R$ (anel), $n = 2^k$, então*

$$\sum_{i=0}^{n-1} a^i = \prod_{i=o}^{k-1} (1 + a^{2^i})$$

Lema 5.1.2. *Seja* $m = \omega^{n/2} + 1$, *onde* $\omega \neq 0 \in \mathbb{Z}$, *então para* $1 \leq p < n$ *temos*

$$\sum_{i=0}^{n-1} \omega^{ip} \equiv 0 \mod m$$

Teorema 5.1.1. *Sejam* n *e* ω *potências de 2 e seja* $m = \omega^{n/2} + 1$, *então* n *tem inverso multiplicativo em* $\mathbb{Z}_m$ *e* ω *é uma raiz n-ésima primitiva da unidade.*

Este teorema nos garante que se p é primo da forma $\omega^{n/2} + 1$, o corpo $\mathbb{Z}_p$ tem ω como n-ésima raiz primitiva, desde que n e ω sejam potências de 2. Portanto, para certos corpos $\mathbb{Z}_p$, é muito fácil encontrar raízes primitivas.
Exemplo.- Tomando $p = 2^4 + 1 = 17$, o corpo $\mathbb{Z}_{17}$ tem 2 como raiz oitava primitiva da unidade.

Outro aspecto importante do teorema anterior é a possibilidade de se computar a transformada rápida de Fourier em corpos finitos, o que não será discutido aqui.

Discutiremos como achar elementos primitivos, isto é, geradores de corpos finitos, ou seja, (q-1)-ésima raiz primitiva da unidade em corpos de cardinalidade q.

Teorema 5.1.2. *No corpo finito* $GF(q)$, ω *é uma* $(q-1)-$*ésima raiz primitiva da unidade se e somente se,*

$$\omega^{(q-1)/a^i} \neq 1 \mod q$$

para todos os divisores primos $a_1, a_2, ..., a_r$ *de* $q - 1$.

Prova. Ver [1]

Teorema 5.1.3. *Seja* ω *uma raiz primitiva de* q. *Então* ω^α *é também uma raiz primitiva se e somente se* $MDC(\alpha, q-1) = 1$.

Prova. ver [1]

Exemplo 5.1.1. *Seja* $\mathbb{Z}_7 = \{0,1,2,3,4,5,6\}$, *os únicos divisores primos de 6 são :* $a_1 = 2, a_2 = 3$. *Verifiquemos o teorema para* 2, $2^3 = 1$,*(falha). Agora podemos provar manualmente que é uma raiz primitiva sexta da unidade. Podemos achar outra raiz usando o teorema 5.1.3. Sabemos que*

$$3^2 = 2; \quad 3^3 = 6; \quad 3^4 = 4; \quad 3^5 = 5; \quad 3^6 = 1,$$

então $\alpha = 5$, *pois* $MDC(5,6) = 1$.*Portanto a outra raiz primitiva é 5. E porque para todo q primo existem* $\phi(q-1)$ *raízes primitivas de q, sabemos que existem só duas raízes primitivas, pois* $\phi(6) = 2$

5.2 O Método de Moenck

Podemos observar que se temos a liberdade de escolher como característica um número primo, então também pode-se escolher corpos que facilitem o apressem a descoberta das raízes dos polinômios .

Em particular, é útil escolher primos p tal que $p-1$ seja altamente composto ($p = L \cdot 2^l + 1$, onde L é pequeno e $l \cong L$). Neste caso, podemos usar a técnica *Divide e Conquista* para refinar o subgrupo multiplicativo de $\mathbb{Z}_p$ que contém uma raiz.

No máximo $l \leq \log p$ de tais refinamentos são necessários para encontrar uma raiz. Isto significa que o tempo para achar as raízes do polinômio é uma função de logaritmo de p, melhor do que p.

Assumimos que as raízes de $g(s)$ são diferentes entre si e não nulos, de outro modo usase o método de Decomposição livre de quadrados, para evitar repetir raízes.

Um caso especial do lema 4.5.1 é que $f(s) = s^{p-1} - 1$ é produto de todos os termos lineares sobre $\mathbb{Z}_p$. Suas raízes são membros do subgrupo multiplicativo $\mathbb{Z}_p^*$,

os membros deste grupo são chamadas $(p-1)$-ésimas raízes da unidade e um gerador do grupo é uma (p-1)-ésima raiz primitiva da unidade.

Note que $f(s) = (s^{(p-1)/2}+1)(s^{(p-1)/2}-1)$, tal que a metade dos $(p-1)$-ésimas raízes da unidades são também $(p-1)/2$-ésimas raízes da unidades. Em geral, $f(s)$ têm $l+1$ fatores da forma:

$$f_i(s) = s^{L\cdot 2^{l-i}} - 1, \quad 0 \leq i \leq l.$$

$f_i(s)$ é o produto de todas as $(p-1)/2^i$-ésimas raízes da unidades de $\mathbb{Z}_p$. Usando este fato, podemos separar as raízes de um polinômio $g(s)$ em raízes que são,

$$r_{i-1}(s) = MDC(g(s), s^{L\cdot 2^{l-i}} - 1)$$

Agora, podemos descrever o processo de refinamento. Seja r_{i-1} um produto de $L \cdot 2^{l-i+1} = (p-1)/2^{i-1}$-ésima raízes da unidade. Então

$$r_i(s) = MDC(r_{i-1}(s), s^{L\cdot 2^{l-i}} - 1) \tag{5.1}$$

é um produto de todas as raízes de $r_i(s)$ que são as $(p-1)/2^i$-ésimas raízes da unidade. Se

$$r_{i-1}(s) = r_i(s) \cdot t_i(s) \tag{5.2}$$

Então $t_i(s)$ é o produto de todas as raízes $r_{i-1}(s)$ que são as $(p-1)\cdot 2^{i-1}-$ ésimas raízes da unidade, mas não $(p-1)\cdot 2^i-$ésimas raízes da unidade. Logo t_i tem a forma:

$$t_i(s) = \prod (s - \omega^{j\cdot 2^{i-1}})$$

onde ω é uma $(p-1)$-ésima raiz da unidade e as j são ímpares.

Se ψ é outra $(p-1)$-ésima raiz da unidade (não necessariamente distinta), então

$$\psi = \omega^m, \quad \text{onde} \quad MDC(p-1, m) = 1$$

Em particular, m é ímpar.

Consideremos formar

$$\begin{aligned} t_i'(s) = \prod(s - \omega^{j \cdot 2^{i-1}} \cdot \psi^{2^{i-1}}) &= \prod(s - \omega^{(j+m)} \cdot 2^{i-1}) \\ &= \prod(s - \omega^{2^i \cdot (j+m)/2}), \end{aligned} \tag{5.3}$$

com j e m ímpares. Assim as raízes de t_i' são $(p - 1/2^i)$-ésima raiz da unidade e o processo de refinamento pode ser aplicado recursivamente a $r_i(s)$ e $t_i'(s)$. Uma vez que as raízes de $t_i'(s)$ são achadas, as raízes de $t_i(s)$ podem ser calculadas dividindo-se por ψ^{2i-1}.

Pode-se expressar a transformação de $t_i(s)$ a $t_i'(s)$ em termos dos coeficientes de $t_i(s)$. Se

$$t_i(s) = \sum_{j=0}^{k} a_j s^j, \quad a_k = 1$$

então expandendo 5.3, vemos que

$$t_i'(s) = \sum_{j=0}^{k} a_j x^j \psi^{(k-j)2^{i-1}}. \tag{5.4}$$

Isto quer dizer que a transformação de $t_i(s)$ para $t_i'(s)$ pode ser executada em k-operações , para dar os coeficientes de $t_i(s)$ e $\psi^{2^{i-1}}$. O procedimento para executar esta transformação será chamado de 'Convert'.

Agora podemos costruir um algoritmo para encontrar as raízes de um polinômio baseados nas relações 5.1, 5.2, 5.4.

5.2.1 *O Algoritmo de Moenck*

Algoritmo 5.1 (Algoritmo: Raízes(r, ψ, i, H).).

Entradas: *(1)* *o polinômio* $r(s)$;

(2) ψ *é uma* $(p-1/2^i)$*-ésima raiz da unidade;*

(3) *um inteiro* i;

(4) H *uma lista dos polinômios da forma:*

$h_i = s^{(p-1)/2^i} - 1 \mod r(s), \quad 0 < i \leq l.$

Saídas: *As raízes de* $r(s)$ *em* $\mathbb{Z}_p$.

Passos:

(1) *Base :* *Se grau(r)=1, então retorne* $\{-r_0\}$;

(2) *Busca Direta:* *De outra maneira, se* $2 \nmid (p-1)/2^i$
então retorne Busca
Direta$(r, \psi, (p-1)/2^i)$;

(3) *Separação de raízes:* *Se não , iniciar*
$g(s) = MDC(r(s), h_i(s))$;
$p(s) = Convert(r(s)/g(s), \psi)$;
$R = \phi$;

(4) *Recursão :* *Se grau*$(g) > 0$
então $R = R \cup$ *Raízes*$(g, \psi^2, i+1, H)$;
se grau$(f) > 0$
então $R = R \cup$ *Raízes*$(f, \psi^2, i+1, H)/\psi$;
Retorne R
fim.

O algoritmo podería ser invocado como

$$R = \text{Raízes}(res, \psi, 1, H);$$

onde ψ é uma $(p-1)$-ésima raiz primitiva da unidade. A operação $\cup$ é a união .

O algoritmo Busca direta, é invocado para encontrar as raízes de $r(s)$ no grupo multiplicativo de ψ por avaliação direta. Como existem só L membros de este grupo e L é escolhido pequeno, então fazer as operações toma pouco tempo.

5.2.2 Custo do Algoritmo

Teorema 5.2.1. *Em um corpo finito $\mathbb{Z}_p$, onde $p = L \cdot 2^l + 1$ e $L \cong l$, as raízes de um polinômio de grau k pode ser calculado em $o(k^2 \log p + k \log^2 p)$ passos*

Prova. Os polinômios $H_i = s^{(p-1)/2^i} - 1 \mod r(s), \quad 1 \leq i \leq l$, podem ser calculados em $o(k^2 \log p)$ passos, onde $k = grau(r)$. Similarmente, cada MDC pode ser calculado em $o(k^2 + k \log p)$ passos. O pior caso ocorre quando o refinamento dos subgrupos não separa todas as raízes . Em este caso o algoritmo de Busca Direta deve ser usado para separar elas. então podem existir como máximo $\log p$ refinamentos, de tal maneira que o custo total é $o(k^2 \log p + k \log^2 p + kL)$. □

5.3 O Método de Rabin

Esta seção tratará a respeito de um algoritmo para achar raízes de polinômios . Uma aplicação desse método é um algoritmo para fatoração que apresentamos logo após.

5.3.1 Achando Raízes em $GF(q)$

Seja $f(x) \in GF(q)$ um polinômio de grau m, queremos achar uma ou todas as raízes de $f(x) = 0$. Para isto usaremos o método implementado devido a Rabin ([28]) o que é probabilístico "in natura"; e é uma geralização do método de Berlekamp para corpos primos $\mathbb{Z}_p$ [5]. A ideia básica é muito simples. Assumindo que q é ímpar, o primeiro passo natural, para achar as raízes de $f(x)$ é partí-lo usando o

cálculo do MDC:

$$f_1(x) = MDC(f(x), x^{q-1} - 1)$$

Se $f_1(x) = 1$ então f não tem raízes em $GF(q)$.
Nesta fase f_1 é da forma:

$$f_1(x) = (x - x_1)(x - x_2)...(x - x_k), \qquad k \leq m.$$

Onde as raízes são diferentes entre se. No seguinte passo usamos a decomposição:

$$x^{q-1} - 1 = (x^d - 1)(x^d + 1), \quad \text{com} \quad d = (q-1)/2.$$

Se para calcular o $MDC(f_1(x), x^d - 1)$ algumas das x_i satisfaz $x_i^d - 1 = 0$ enquanto que as outras satisfazem $x_i^d + 1 = 0$, então o MDC será um divisor não trivial de $f_1(x)$, e com isto temos logrado achar um fator $x - x_i$, isto é uma raiz, de $f(x)$. Em geral ao calcular o $MDC(f_1(x), x^d - 1)$, ninguém garante que este MDC vai ser diferente de 1, ou de $f_1(x)$, mas a contribução de Rabin é mostrar que esta situação pode ser evitada por aleatoriedade. Um incremento t é aleatoriamente escolhido em $GF(q)$ e adicionado a x no último cálculo do MDC, onde $(x^d - 1)$ é sustituido por $((x+t)^d - 1)$. A prova é dada em [28]. O método resultante é prontamente trasladado no algoritmo seguinte:

5.3.2 O Algoritmo de Rabin para achar raízes em GF(q)

(1) Seja $f_1(x) = MDC(f(x), x^{q-1} - 1)$; se o grau de f_1 é zero, então retorna negativo.

(2) Escolhemos t aleatoriamente em GF(q);

repetir

$$f_t = MDC(f_1(x), (x+t)^d - 1), \qquad \text{onde} \quad d = (q-1)/2$$

até que $f_t \neq 1$ e $f_t \neq f_1$.

(3) Seja

$$f_2(x) = \begin{cases} f_t(x), & \text{se grau}(f_t) \leq \text{grau } (f_1)/2 \\ f_2(x)/f_t(x), & \text{o.c.} \end{cases}$$

Se f_2 é linear, então retornamos f_2.

(4) $f = f_2$, vai para passo (2).

Quando q é par, no passo (2), $((x+t)^d - 1)$ é substituido por $x + x^2 + \ldots + x^{2^{n-1}}$.

5.3.3 O custo do Algoritmo

O número de operações aritméticas necessárias para achar f_1 e f_2 é $o(n.m\ L(m) \log\ p)$. Como $\partial(f_2) \leq 1/2m$, temos que o número de operações para achar f_3 é no máximo a metade do número de operações para achar f_2; e assim sucessivamente. No total o número de operações usadas para achar as raízes de $f(x)$ é $o(nmL(m) \log\ p)$.

Em termos de operações em $\mathbb{Z}_p$, cada operação requer $o(nL(m))$ operações com resto módulo p. Assim o número de operações em $\mathbb{Z}_p$, para achar raízes é

$$o(n^2 m\ \log\ (m) \log\ (p)\ L(n)),$$

onde $L(n) = \log(n)\ \log(\log\ n)$, pode-se ver que o maior tempo é gasto durante as exponenciações.

5.4 Uma Aplicação do Método de Rabin para Fatorar Polinômios em GF(p)

Embora os métodos probabilísticos não tem provados por si mesmos ser tão eficientes quanto os determinísticos, eles tem sido implementados por várias razões. Uma de elas é que podem ser usados para designar modelos em extensão de corpos. Esta é a causa pela qual o método de Rabin, ver [28], que sempre é mais caro que o de Cantor-Zassenhaus; ver[15], é estudado aqui. *O Metodo de Rabin* faz uso do algoritmo para achar raízes da secção anterior , e pode ser dado da seguinte maneira. Seja $f(x)$ com grau n, sobre $GF(p)$, seja $g(x)$ o MDC, onde

$$g_m(x) = MDC(f(x), x^{p^m} - x), \qquad 1 \leq m < n$$

Pelo algoritmo de Cantor-Zassenhaus, sabemos que $g_m(x)$ é o produto de todos os fatores irredutíveis $h(x)|f(x)$ de grau $i|m$. Portanto tem-se que achar $g_m(x) \neq 1$ com o menor m possível. Seja grau(g)=l. Sabemos que $g_m(x)$ é da forma

$$g_m(x) = h_1(x)...h_k(x) \quad (k.m = l)$$

onde h_i é irredutível e de grau m.

Neste caso é suficiente encontrar uma raiz a de $g_m(x) = 0$ em $GF(p^m)$. Esta será a raiz de um único h_i. O próximo passo é a reconstrução de h_i.

Rabin dá dois métodos para alcançar esta meta. Selecionamos um deles, que consiste em calcular em $GF(p^m)$ a expresão

$$h_i(x) = (x - a)(x - a^p)...(x - a^{p^{m-1}})$$

o correspondiente algoritmo esta dirigido nesse sentido. A é o polinômio para fatorar.

5.4.1 *O Algoritmo de Rabin para Fatorar em GF(q)*

(1) (Fatoração de grau diferente).Seja m=grau(A)/2; escolher o primeiro $l \in \{1, ..., m\}$ tal que

$$g_l = MDC(A(x), x^{p^l - 1}) \neq 1$$

do contrario retorne.

(2) (Agora

$$g_l(x) = \prod_{i=1}^{k} h_i(x), grauh_i = ... = grauh_k = l.)$$

Encontrar uma raiz a de $g_l \in GF(p^l)$. O correspondente fator $h_i(x)$ está dado por

$$h_i(x) = (x - a)(x - a^p)...(x - a^{pl-1}).$$

(3) Juntar os fatores h_l, $A = A/h_l$, volte para (1)

5.4.2 *O custo do Algoritmo*

O número total de operações em $\mathbb{Z}$ para fatorar um polinômio $f(x) \in \mathbb{Z}_p$ de grau n é:

$$o(n^3 \log n \ L(n) \log \ p) + o(n^3 \ L(n)^2 \log \ p) + o(n^3).$$

Aqui estão incluídas as operações necessárias para achar um polinômio irredutível $g_i(x)$ o que vai dar origem ao corpo $GF(p^m)$. O último termo representa as operações usadas para resolver equações lineares usando o algoritmo de polinômios irredutíveis.

6 CONSIDERAÇÕES FINAIS

Neste trabalho apresentamos corpos finitos como um importante sistema algébrico que é adequado para a álgebra computacional. Caracterizamos os corpos finitos apresentando, sua estrutura, as possíveis cardinalidades, a existência, a unicidade e a sua construção . Nossa principal preocupação foi a aritmética efetuada nos corpos finitos e, como conseqüencia a sua eficiência. Estudamos dois problemas (que são importantes por si mesmo, mas que têm outras aplicações): A fatoração e cálculo de raízes de polinômios .

Concluímos que usando as rotinas apresentadas aqui, podem-se achar os fatores irredutíveis de polinômios , assim como suas raízes. A implementação é feita principalmente para dar os exemplos e não para comparar eficiências de tempos de corrida.

Estes são os principais pontos que identificamos em nosso trabalho.

- Para fatoração de polinômios sobre corpos finitos, em primeiro lugar fatoramos sobre corpos com característica p pequena ($p < n$), usando o algoritmo de Berlekamp, que é determinístico.
- Quando fatoramos sobre corpos finitos grandes usamos o método de Cantor Zassenhaus.
- Para fatorar sobre corpos de Galois $GF(p^k)$, $k > 1$ cujos coeficientes estão em $GF(p)$, usamos o método de Rabin.
- Dados um corpo finito com uma característica especial da forma $p = L \cdot 2^l + 1$, onde $L \cong l$ podemos fatorar um polinômio , usando o método de Moenck, que consiste em reduzir o passo final do algoritmo de Berlekamp [6] ao problema de achar raízes de um polinômio em um corpo $\mathbb{Z}_p$.

- Para achar raízes temos os algoritmos de Moenck e de Rabin, onde para uma característica especial $p = L \cdot 2^l + 1$, o de Moenck é melhor que o de Rabin.

Quanto aos tempos computacionais dos diferentes algoritmos estudados neste trabalho, façamos as comparações teóricas.

O algoritmo de Berlekamp faz no pior caso, $o(n^3 + r\ p\ n^2)$ operações em $GF(p)$, onde r é o número de fatores irredutíveis, $r \cong log\ n$, n é o grau do polinômio . Note que o tempo de corrida é proporcional a p, o que não é muito bom, quando p é grande. Neste caso o mesmo Berlekamp aconselha usar um outro algoritmo.

O tempo de corrida do Algoritmo de Cantor Zassenhaus é polinomial em n, o grau do polinômio e $log\ q$, sendo q a cardinalidade do corpo, o que faz este procedimento melhor que o de Berlekamp. A desventagem esta no fato em que estes algoritmos precisam executar a aritmética dentro do corpo, o que é bastante custoso quando se trata de uma extensão de $\mathbb{Z}_p$.

O problema de fatorar mais comúm é quando um polinômio $f(x)$ com coeficientes em $\mathbb{Z}_p$ vai ser fatorado sobre $GF(p^k)$; podemos acelerar o método de Cantor-Zassenhaus fatorando em primeiro lugar, sobre $GF(p)$ e logo sobre $GF(p^k)$ aqueles fatores que são irredutíveis sobre $\mathbb{Z}_p[x]$. O seguinte resultado usa-se sem custo algúm para obter o número de fatores irredutíveis e seus respeitivos graus em $GF(p^k)$.

Teorema 6.0.1. *Seja f um polinômio irredutível de grau n em $GF(q)$. Se k é natural e $d = MDC(n,k)$ então f tem d fatores irredutíveis sobre $GF(q^k)$ todos de grau n/d.*

Observe que quando n e k são primos, o fator já é irredutível sobre a extensão e não é preciso parti-lo. Por Exemplo, se temos o polinômio irredutível,

$$p(x) = x^4 + 1 \ \in \ GF(3)$$

Suponha que $k = 6$ e $d = 2 = MDC(4, 6)$, logo pelo teorema anterior temos que o polinômio têm 2 fatores em $GF(3^6)$ de grau $2 = n/d$. Ditos fatores podem-se achar usando a implementação do algoritmo de Rabin.

BIBLIOGRAFIA

[1] R.B.J.T. Allenby and E.J.Redfern, *Introduction to Number Theory with Computing*, Routledge, Chapman and Hall, Inc., New York, 1989.

[2] M.Ben-Or, *Probabilistic Algorithmis in Finite Fields*, Proc22nd IEEE Symp. Found. Comp. Sci.,1981, pp.394-398.

[3] Bhubaueswer Mishra. *Algorithmic Algebra.* Springer-Verlag, New York.Inc,1993.

[4] E.R.Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, 1968.

[5] E.R.Berlekamp, *Factoring polynomials over finite fields*, Bell System Tech.J., vol.46,1967, pp.1853-1859.

[6] E.R.Berlekamp, *Factoring polynomials over large finite fields*, Math.Comp., vol.24,1979, pp.713-735.

[7] G.Brassard,P.Bratley.*Algorithmics, Theory and Practice.*Prentice-Hall.Inc.New Yersey.1988.

[8] B.W.Char, K.O.Geddes, G.H.Gonnet,*MAPLE V: Language Reference Manual*, Springer-Berlag, New York, Inc., 1991.

[9] B.W.Char, K.O.Geddes, G.H.Gonnet,*First Leaves: A Tutorial Introduction to MAPLE V* , Springer-Berlag, New York, Inc., 1992.

[10] Darren Redfern, *The MAPLE Handbook*, Springer-Berlag, New York, Inc., 1993-1994.

[11] B.Buchberger,G.E.Collins, and R.Loos.*Computer Algebra, Simbolic and Algebraic Computation*, Springer-Verlag, New York, 1983.

[12] J.Calmet, *Algebraic Algorithms in GF(q)*, Discrete Mathematics, No.56, vol.1, 1985, pp.101-109.

[13] J.Calmet, *Deterministic versus Probabilistic Factorization of Integral Polynomials*, PROCEEDINGS EUROCAM'82, LNCS, pp.117-125.

[14] J.Calmet and R.Loos, *An inprovement of Rabin's probabilistic algorithm for general irreducible polynimials over GF(p)*, Inf.Proc.Letters, Vol.11, No.2, Oct.1980, pp.94-95

[15] D.Cantor and H.Zassenhaus,*A New Algorithm for factoring polynomials over finite fields*, Math. Comp., 36 (1981), pp. 587-592.

[16] G.E.Collins, M.Mignotte and F.Winkler,*Arithmetic in basic algebraic domains*, in Computer Algebra-Symbolic and Algebraic Computation, Computing, Suppl.4, Springer-Verlag, 1982, pp. 189-220.

[17] J.H. Davenport, Y. Siret and E. Tournier.*Computer Algebra: Systems and Algorithms for Algebraic Computation* Academic press, England, 1988.

[18] André Heck, *Introduction to Maple*, Springer-Verlag.New York.Inc.1993.

[19] I.N.Herstein, *Topics in Algebra*, Jhon Wiley Sons.Inc., 1975.

[20] E. Kaltofen, *Factorization of Polynomials*, in Computer Algebra-Symbolic and Algebraic Computation, Computing, Suppl.4, Springer-Verlag, 1982, pp. 95-113.

[21] E. Kaltofen, *Polynomial Factorization 1982-1986*, Department of Computer Science, RPI, No.86-19, Renselaer Polytechnic Institute, Troy, New York, 1986.

[22] D.E.Knuth, *The Art of Computer Programming,*vol.2:*Seminumerical Algorithms,*Addison-Wesley, Reading,Mass.,USA, 1969.

[23] D.Lazard, *On Polynomial Factorization*, LNCS, Proc. EUROCAM'82, pp.127-134.

[24] M.Lauer, *Computing by Homomorphic Images*, U. Karlsruhe, Germany, 1982.

[25] John D.Lipson, *Elements of Algebra and Algebraic Computations*, Addison-Wesley, 1981.

[26] R.T.Moenck, *On the Efficiency of Algorithms for Polynomial Factoring*, Math.Comp., 31, No.137,1977, pp.235-250.

[27] M.Pohst and H. Zassenhaus,*Algorithmic Algebraic Number Theory,* Cambridge University Press, 1989, Cambridge, England.

[28] M.O.Rabin, *Probabilistic Algorithms in Finite Fields.*SIAM J. Comput.,9,1980, pp.273-288.

[29] V.M.Rodrigues, *Algoritmos para o Máximo Divisor Comúm de Polinômios a uma variável*, Dissertação de Mestrado, CPGMAP-UFRGS, 1995.

[30] V.Trevisan, and P. S. Wang,*Practical Factorization of univariante polynomials over finite fields,* Proceedings of ISSAC'91, Bonn.

[31] V.Trevisan, *Computação Algébrica e Simbólica,* Boletim da SBMAC, Vol.2, serie 2, 1990.

[32] V.Trevisan, *Univariante Polynomials Factorization*, Kent State University Graduate College, Kent, Ohio, USA, 1992.

[33] V.Trevisan,*Polynomial Factorization*, Mat.Comp., Nro.7, pp.185-199, 1994.

[34] B.L. Van Der Waerden. *Modern Algebra.* Vol.1. 2da.rev. Springer-Verlag. New York. 1949.

[35] H.Zassenhaus, *Polynomial time factoring of integral polynomials*, SIGSAM Bulletin, Vol. 15, No.2, 1981, pp.6-7.

ANEXO A-1 IMPLEMENTAÇÃO DOS ALGORITMOS

Neste capítulo apresentamos os exemplos elaborados em MAPLEV.3. Ambos programas precisam ter como entradas um polinômio em $GF(q)$ e um número positivo primo como módulo. E teremos com saídas os fatores irredutíveis.

A-1.1 Algoritmo NDDF

Usamos os mesmos exemplos em diferentes módulos.

```
> pol;
```

$$x^8 + x^6 + 10\,x^4 + 10\,x^3 + 8\,x^2 + 2\,x + 8$$

```
> nddf(pol,13);
```

$$[x + 3, 1, x^3 + 8\,x^2 + 4\,x + 12, x^4 + 2\,x^3 + 3\,x^2 + 4\,x + 6]$$

```
> nddf(pol,19);
```

$$[x + 3, x^4 + 8\,x^3 + 7\,x^2 + 2\,x + 9, x^3 + 8\,x^2 + 15\,x + 1]$$

```
> nddf(pol,19);
```

$$[x + 3, x^4 + 8\,x^3 + 7\,x^2 + 2\,x + 9, x^3 + 8\,x^2 + 15\,x + 1]$$

```
> nddf(pol,31);
```

$$[x^2 + 5\,x + 13, x^2 + x + 18, x^4 + 25\,x^3 + x^2 + 14\,x + 26]$$

$$pol1 := x^{12} + 61\,x^{11} + 20\,x^{10} + 31\,x^9 + x^8 + 69\,x^7 + 4\,x^6 + 55\,x^5 + 32\,x^4 + 7\,x^3 + 49\,x^2 + 52\,x + 46$$

```
> nddf(pol1,13);
```

$$[x^3 + 5\,x^2 + 4\,x + 9, x^3 + x^2 + 4, x^6 + 3\,x^5 + 6\,x^4 + 3\,x^3 + 5\,x^2 + 2\,x + 2]$$

```
> nddf(pol1,19);
```

$$[x + 8, 1, 1, x^4 + 15\,x^3 + 13\,x^2 + 15\,x + 9, x^7 + x^5 + 3\,x^4 + 12\,x^3 + x^2 + 8\,x + 17]$$

```
> nddf(pol1,31);
```

$$[x + 19, 1, 1, x^4 + 11\,x^3 + 12\,x^2 + 10\,x + 29, x^7 + 16\,x^5 + 26\,x^4 + 23\,x^3 + 16\,x + 20]$$

$$pol2 := x^9 + 12\,x^8 + 69\,x^7 + 48\,x^6 + 11\,x^5 + .6\,x^4 + 2\,x^3 + 24\,x^2 + 45\,x + 3$$

```
> nddf(pol2,19);
```

$$[x + 9, 1, 1, x^8 + 3\,x^7 + 4\,x^6 + 12\,x^5 + 17\,x^4 + 5\,x^3 + 14\,x^2 + 12\,x + 13]$$

```
> nddf(pol2,23);
```

$$[x + 18, 1, 1, x^8 + 17\,x^7 + 16\,x^6 + 13\,x^5 + 7\,x^4 + 18\,x^3 + x + 4]$$

```
> nddf(pol2,31);
```

$$[x^3 + 27\,x^2 + 30\,x + 11, x^2 + 5\,x + 23, x^4 + 11\,x^3 + 25\,x^2 + 25\,x + 13]$$

A-1.2 Algoritmo de Cantor- Zassenhaus

Da mesma maneira, também usamos diversos módulos.

```
> canzas(pol,23);
```

pol;

$$x^8 + x^6 + 10\,x^4 + 10\,x^3 + 8\,x^2 + 2\,x + 8$$

$$[x^2 + 16\,x + 20, x^6 + 7\,x^5 + 7\,x^4 + x^3 + 15\,x^2 + 3\,x + 5]$$

```
> canzas(pol,5);
```

$$[x + 4, x^3 + 2\,x + 4, x^4 + x^3 + x + 3]$$

```
> canzas(pol,31);
```

$$[x + 17, x + 19, x^2 + x + 18, x^4 + 25\,x^3 + x^2 + 14\,x + 26]$$

```
> canzas(pol,19);
```

$$[x + 3, x^2 + 4\,x + 9, x^2 + 4\,x + 1, x^3 + 8\,x^2 + 15\,x + 1]$$

$$pols := x^9 + 2\,x^8 + 2\,x^6 + 2\,x^4 + 15\,x^3 + 8\,x^2 + 5\,x + 11$$

```
> canzas(pols,11);
```

$$[x, x + 6, x^3 + 10\,x^2 + 10\,x + 5, x^4 + 8\,x^3 + 4\,x + 2]$$

```
> canzas(pols,13);
```

$$[x+3, x+9, x^7+3x^6+2x^5+x^4+12x^3+3x+11]$$

```
> canzas(pols,23);
```

$$[x+22, x+21, x^2+7x+20, x^2+20x+1, x^3+x^2+9x+2]$$

```
> canzas(pols,19);
```

$$[x+5, x^8+16x^7+15x^6+3x^5+4x^4+x^3+10x^2+15x+6]$$

```
> canzas(pols,31);
```

$$[x+13, x^8+20x^7+19x^6+3x^5+23x^4+13x^3+x^2+26x+8]$$

A-1.3 Algoritmo para achar raízes

Este algoritmo encontra uma raiz em $GF(p)$.

$$pol := x^7+3x^5+6x^2+x^6+9x^4+6x+11x^3+3$$

```
> pol;
```

$$x^7+3x^5+6x^2+x^6+9x^4+6x+11x^3+3$$

```
> getroot(pol,5);
```

$$x+11$$

```
> getroot(pol,5);
```

$$x + 3$$

```
> Ra:=
```

$$x^{12} + 10\,x^{11} + 9\,x^{10} + 17\,x^9 + 18\,x^8 + 22\,x^7 + 14\,x^6 + 7\,x^5 + 21\,x^4 + 10\,x^3 + 9\,x^2 + 11\,x + 13$$

```
> getroot(Ra,13);
```

polinomio sem fatores lineares

```
> getroot(Ra,11);
```

polinomio sem fatores lineares

$$pol1 := x^{11} + 8\,x^6 + 13\,x^8 + 3\,x^3 + 16\,x^9 + 9\,x^4 + 12\,x + 6\,x^{10} + 2\,x^5 + 8\,x^7 + 16\,x^2 + 3$$

```
> getroot(pol1,13);
```

$$x + 7$$

```
> getroot(pol1,13);
```

$$x + 6$$

```
> getroot(pol1,13);
```

$$x + 1$$

```
> getroot(pol1,19);
```

$$x + 7$$

```
> getroot(pol1,19);
```

$$x + 6$$

```
> getroot(pol1,31);
```

$$x + 7$$

$$pol2 := x^{16} + 2\,x^{14} + 9\,x^9 + 5\,x^7 + 6\,x^{11} + 16\,x^4 + 15\,x^2 + 13\,x^{15} + 6\,x^{13} + 14\,x^8 + 16\,x^6 + 10\,x^{10} + 4\,x^3 + 11\,x + 16\,x^{12} + 3\,x^5 + 1$$

```
> getroot(pol2,17);
```

$$x + 5$$

```
> getroot(pol2,17);
```

$$x + 11$$

A-1.4 Algoritmo de Rabin

Este Algoritmo encontra os fatores em $GF(p^K)$.

```
> pol;
```

$$x^8 + x^6 + 10\,x^4 + 10\,x^3 + 8\,x^2 + 2\,x + 8$$

```
> rabin(pol,31);
```

$$[x - 12, x - 14, x^2 + x + 18, x^4 + 25\,x^3 + x^2 + 14\,x + 26]$$

$$juno := 20\,x^7 + 22\,x^6 + 18\,x^5 + 3\,x^4 + 10\,x^3 + 19\,x^2 + x + 7$$

```
> R:=rabin(juno,23);
```

$$R := [x + 2,\ x^2 + 20\,x + 17, 20\,x^4 + 19\,x^3 + x^2 + 12\,x + 9]$$

$$pol1 := x^{13} + 10\,x^{12} + 17\,x^{11} + 9\,x^{10} + 19\,x^9 + 30\,x^8 + 8\,x^7 + 21\,x^6 + 26\,x^5 + 16\,x^4 + 5\,x^3 + 2\,x^2 + 2\,x + 2$$

```
> rabin(pol1,11);
```

$$[x - 4, x - 7,\ x^4 + 7\,x^3 + 9\,x^2 + 4\,x + 2,\ x^7 + 3\,x^6 + 3\,x^5 + 7\,x^4 + 6\,x^3 + 4\,x^2 + 9\,x + 2]$$

$$pol2 := x^{10} + 2\,x^8 + 76\,x^7 + 7\,x^6 + 66\,x^5 + 28\,x^4 + 24\,x^3 + 32\,x^2 + 17\,x + 47$$

```
> rabin(pol2,41)
```

$$(x^2 + 26\,x + 16)(x^2 + 18\,x + 28)(x^4 + 34\,x^3 + 4\,x^2 + 15\,x + 19)(x^2 + 4\,x + 15)$$

ANEXO A-2 CÓDIGO MAPLE PARA FATORAR POLINÔMIOS

Estes programas foram elaborados em código MAPLEV.3.

Programa Principal para Fatorar segundo Cantor-Zassenhaus.

```
with(linalg):

canzas :=

   proc(pol,modo)
   local i,n,j,h,rlist,czfac;
       h := nddf(pol,modo);
       n := 1;
       for i in indices(h) do
           if degree(h[i[1]]) = i[1] then
               rlist[n] := h[i[1]]; n := n+1
           elif degree(h[i[1]]) <> 0 then
               czfac := sepcz(h[i[1]],i[1],modo);
               for j to nops(czfac) do  rlist[j+n] := czfac[j] od;
               n := j+n
           fi
       od;
       RETURN(convert(eval(rlist),list))
   end
```

Subprograma para obter todos os fatores de grau diferente.

```
nddf :=

  proc(pol,modo)
  local i,n,r,h,g,Q,rest,rlis,variax,f,tmp,rli,T,ind,q;
      f := pol;
      q := modo;
      n := degree(f);
```

```
        Q := array(1 .. n,1 .. n);
        i := 1;
        r := Rem(x^q,f,x) mod q;
        g := f;
        while i < n do
            h[i] := Gcd(g,r-x) mod q;
            print(i);
            if g = h[i] then i := n+1
            else
                g := Quo(g,h[i],x) mod q;
                i := i+1;
                if degree(g) <= 2*i then h[degree(g)] := g; i := n+1
                else
                    Q := forma(f,q);
                    rest := getcoef(r,n);
                    print(Q);
                    rest := convert(rest,matrix);
                    tmp := evalm(transpose(rest) &* Q);
                    print(tmp);
                    rli := map(modp,tmp,q);
                    rlis := convert(rli,vector);
                    r := getpol(rlis)
                fi
            fi
        od;
        T := convert(eval(h),list);
        ind := nops(T);
        RETURN(h)
    end
```

Subprograma para obter coeficientes.

```
getcoef := proc(r,deg)
           local rest,i;
               rest := array(1 .. deg);
               for i to deg do  rest[i] := coeff(r,x,i-1) od;
               RETURN(eval(rest))
```

```
        end
```

Subprograma para formar a matriz de Berlekamp.

```
forma :=

    proc(f,q)
    local n,i,j,k,Q;
        n := degree(f);
        Q := array(1 .. n,1 .. n);
        for i to n do
            for j to n do
                Q[i,j] := coeff(rem(x^((i-1)*q),f,x) mod q,x,j-1)
            od
        od;
        evalm(Q)
    end
```

Subprograma para obter um polinômio como resultado da multiplicação da matriz de Berlekamp por um vetor.

```
getpol := proc(v)
          local r,j,vari;
              r := vectdim(v);
              vari := vector(r,[vari[j]]);
              for j to r do  vari[j] := x^(j-1) od;
              print(vari);
              r := dotprod(v,vari);
              print(r);
              RETURN(r)
          end
```

Subprograma para separar um polinômio, dado por o NDDF, em fatores do mesmo grau.

```
sepcz :=

    proc(pol,i,p)
    local h2,t,g,t1,l1,l2,h1,l,m,c0,lt,fa;
        h1 := pol;
        l := degree(h1,x);
        m := l/i;
        c0 := 0;
        while c0 < m do
            l2 := rand(1 .. 2*i-1);
            lt := l2();
            t := Randpoly(lt,x) mod p;
            t1 := Expand(t^(1/2*p^i-1/2)) mod p;
            t := Rem(t1,h1,x) mod p;
            g := Gcd(h1,t-1) mod p;
            if h1 <> g and degree(g,x) <> 0 then
                h1 := Quo(h1,g,x,'r') mod p;
                if degree(h1,x) = i then
                    c0 := c0+1; fa[c0] := h1; h1 := h2
                fi;
                if degree(g,x) = i then c0 := c0+1; fa[c0] := g
                else h2 := g
                fi
            fi
        od;
        fa := convert(eval(fa),list);
        RETURN(fa)
    end
```

Programa Principal para Fatorar segundo Rabin.

```
rabin :=

proc(pol,p)
local L,i,ind,h,f,k,k1,k2,numk,fak,conta,g,m,j;
    L := nddf(pol,p);
```

```
    L := convert(eval(L),list);
    ind := nops(eval(L));
    i := 1;
    conta := 1;
    fak[1] := factlin(L[1],p);
    fak[1] := convert(eval(fak[1]),list);
    i := 2;
    conta := degree(L[1],x)+1;
    while i <= ind do
        h := L[i];
        m := degree(h,x);
        if m = 0 then i := i+1
        elif m = i then
            fak[conta] := h; conta := conta+1; i := i+1
        else
            f := Randprime(i,x) mod p;
            alias(a = RootOf(f));
            k := Roots(h,a) mod p;
            numk := m/i;
            if k = [] then fak[conta] := h
            else
                j := 1;
                while j <= numk do
                    k1 := k[j];
                    k2 := k1[1];
                    fak[conta] :=
                        Expand(product(x-k2^(p^(l-1)),l = 1 .. i))
                         mod p;
                    conta := conta+1;
                    j := j+1
                od
            fi;
            i := i+1
        fi
    od;
    fak := convert(eval(fak),list);
    RETURN(fak)
end
```

Subprograma encarregado de sacar os fatores lineares.

```
factlin :=

    proc(pol,p)
    local h,m1,fak,k,i,l;
        m1 := degree(pol,x);
        if 1 < m1 then
            k := Roots(pol) mod p;
            for i to m1 do  l := k[i]; l := l[1]; fak[i] := x-l od
        fi;
        if m1 = 1 then fak[1] := pol fi;
        RETURN(eval(fak))
    end
```