ASPECTOS ALGÉBRICOS E COMPUTACIONAIS DA TRANSFORMADA RÁPIDA DE FOURIER

(Vilmar Trevisan)

Essa dissertação é um requisito parcial para a obtenção do título acadêmico de Mestre no Curso de Pós-Graduação em Matemática da UFRGS. O trabalho foi orientado pelo Dr. Julio Cesar Ruiz Clayssen e teve suporte financeiro parcial do CNPq.

Defendida em 12.08.1986

AGRADECIMENTOS

Ao professor Julio pelo apoio e

para a Eliana, pelo estímulo sempre presente.

RESUMO

A Transformada Rápida de Fourier (FFT) é apresentada como um algoritmo que calcula a Transformada Discreta de Fourier mais eficientemente, do ponto de vista computacional.

Uma versão mais moderna do algoritmo de Cooley e Tukey é considerada com a finalidade de se obter aplicações da FFT em algoritmos puramente algébricos, como operações com polinômios e a multiplicação de inteiros. Nas aplicações em questão, são levados em conta os aspectos computacionais e algumas implementações são apresentadas.

SUMÁRIO

INTRODUÇÃO

O fato de que a transformada rápida de Fourier (FFT) aparece freqüentemente nos mais diversos campos, estando fortemente vinculada à Álgebra Computaconal (Auslander), (Lehmer), (Gentleman), (Bergland), (Cooley), (Borodin), (Lipson), foi um dos motivos que nos levaram à elaboração deste trabalho, que trata principalmente da parte algébrica e aplicações da FFT.

No capítulo I, fazemos breve comentário da transformada discreta de Fourier (TDF). Para leituras adicionais, recomenda-se (Elliot), (Henrici), (Arsac).

O capítulo II trata da FFT. É apresentado um algoritmo FFt em versao mais moderna do original de (Coley-Tukey). O capítulo VI traz um programa em linguagem PASCAL que calcula a TDF via FFT.

As operações com polinômios via um esquema de avaliação-interpolação e auxílio da FFT é objeto de estudo do capítulo III. Uma complementação desse assunto pode ser vista em (Aho), (Lipson), (Borodin). A implementação de alguns algoritmos deste capítulo estão no último capítulo.

No capítulo IV, tratamos da viabilidade da FFT em corpos finitos Zp, enquanto que o capítulo VI apresenta um programa que calcula um elemento primitivo de Zp.

Outra aplicação algébrica da FFT é estudada no capítulo V: Amultiplicação deinteiros muito grandes. Para tal é necessária

e o problema chinês do resto (PCR) é estudado e resolvido (Aho), (Borodin), (Pollard), (Lipson).

Finalmente o capítulo VI traz algumas implementações computacionais dos capítulos anteriores. Todos os programas são em linguagem PASCAL e foram rodados no micro computador Nexus 1600.

CAPÍTULO I

A TRANSFORMADA DISCRETA DE FOURIER

1.INTRODUÇÃO:

Analogamente a teoria usual de expansão de em séries de Fourier, que está baseada no conhecimento da função em um intervalo, existe uma teoria que é baseada no conhecimento de uma função em um determinado conjunto discreto de pontos igualmente espaçados, a qual é de importância computacional muito grande.

A operação central na análise numérica de Fourier é a formação das somas

$$A_k = \sum_{j=0}^{n-1} X_k \, W^{-kj}, \qquad k=0,\ldots,n-1,$$

onde n é um inteiro positivo, $X_0,\ldots,X_{n-1}$ são escalares (reais ou complexos) e $W=e^{2\,i/n}$, é uma n-raiz primitiva da unidade.

Estas somas ocorrem como aproximação dos coeficientes de Fourier, na construção de polinômios trigonométricos de interpolação, na solução de problemas de contorno, em equações em diferenças, como aproximação da transformada integral de Fourier e, inclusive como transformação de coordenadas em equações diferenciais parciais (Henrici), (Bergland).

Neste capítulo apresentaremos a transformada de Fourier de maneira operacional, isto é, como uma transformação linear em um espaço de dimenção finita e caracterizada por propriedades peculiares de periodicidade.

2. A MATRIZ DE FOURIER:

Seja $n>0$ um inteiro fixo e fazemos

$$(1)\quad W=e^{2\pi/n}=\cos 2\pi/n + i\,\mathrm{sen}\, 2\pi/n.$$

Chamamos de **matriz de Fourier de ordem n** a matriz

$$(2)\quad F=n^{-1/2}(W^{-ij}),\ i,j=0,1,\ldots,n-1.$$

Dado que a seqüência W^{k}, $k=0,1,\ldots,n,\ldots$ é periódica de período n; então só existem n elementos distintos em F. Em particular se F* denota a transposta conjugada da matriz F, então podemos escrever (3) alternativamente como segue.

$$(3)\quad F^{*}=n^{-1/2}(W^{ij})=n^{-1/2}\begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & W & W^{2} & \ldots & W^{n-1} \\ 1 & W^{2} & W^{4} & \ldots & W^{2(n-1)} \\ \vdots & & & & \\ 1 & W^{n-1} & W^{n-2} & \ldots & W^{(n-1)} \end{bmatrix}$$

Se a matriz dos expoentes E é dada por

$$(4) \qquad E = \begin{bmatrix} 0 & 0 & 0 \ldots 0 & 0 \\ 0 & 1 & 2 \ldots n-2 & n-1 \\ 0 & 2 & 4 \ldots n-4 & n-2 \\ \vdots & & & \\ 0 & n-2 & n-4 \ldots 4 & 2 \\ 0 & n-1 & n-3 \ldots 2 & 1 \end{bmatrix} = (kj \pmod n)$$

E, dessa forma, temos

$$(5) \qquad F^* = n^{-1/2} W^E = n^{-1/2} \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & W & W^2 & \ldots & W^{n-1} \\ 1 & W^2 & W^4 & \ldots & W^{n-2} \\ \vdots & & & & \\ 1 & W^{n-1} & W^{n-2} & \ldots & W \end{bmatrix}$$

É de fundamental importância que F é unitária, ou seja,

$$F.F^* = F^*.F = I,$$

significando que F* é a inversa da matriz de Fourier F. Esse resultado segue da identidade geométrica

$$(6) \qquad \sum_{r=0}^{n-1} W^{r(j-k)} = \begin{cases} n, & \text{se } j=k \\ 0, & \text{se } j \neq k \end{cases}$$

que será demonstrada na proposição 1 do capítulo seguinte.

Uma segunda aplicação da identidade (6) estabelece que

$$(F^*)^2 = F^2 = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 1 \\ \vdots & & & & \\ 0 & 0 & 1 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \end{bmatrix}$$

Em particular, obtemos que $F^4 = (F^*)^4 = I$. Portanto, os autovalores de F são 1, -1, i, -i, com multiplidades apropriadas.

3.A TRANSFORMADA DISCRETA DE FOURIER:

Trabalhando com n-uplas complexas, escreveremos as matrizes coluna

$$X = (X_0, \dots, X_{n-1})$$
$$A = (A_0, \dots, A_{n-1})$$

A transformação linear

$$(7) \qquad A = n^{1/2} F X,$$

onde F é a matriz de Fourier, é conhecida como a **trasformada discreta de Fourier**. Sua inversa é dada simplesmente por

$$(8) \qquad X = n^{-1/2} F^* A.$$

Em termos de coordenadas, temos

$$(7') \qquad A_r = \sum_{k=0}^{n-1} X_k W^{-kr}, \qquad r = 0, \dots, n-1$$

$$(8') \qquad X_k = 1/n \sum_{r=0}^{n-1} A_r W^{kr}, \qquad k = 0, \dots, n-1.$$

Essas relações são particularmente úteis com dados periódicos, pois observa-se que $W^{(r+pn)k} = W^{rk}$, para qualquet p inteiro, de modo que $A_{r+pn} = A_r$ e $X_{k+pn} = X_k$.

Quando as seqüências (X_k) e (A_r) são periódicas de período n (para r,k > 0 e r,k < 0), a notação matricial (7) e (8) pode ser mantida sm amnbigüidade: se $X = (X_k)$ é uma seqüência de período n, denotamos por $A = (A_r) = n^{1/2} F X$ a seqüência periódica cujo r-ésimo termo é dado por (7') e analogamente a inversa (8).

Convolução:

Se $X = (X_k)$ e $Z = (Z_k)$ são duas seqüências periódicas, definimos o produto de convolução de X e Z como sendo a seqüência U = X*Z, com elementos

$$U_r = \sum_{k=0}^{n-1} X_k Z_{r-k}.$$

Exemplo 1: A multiplicação d polinômios e também a multiplicação de inteiros pode ser executada através da convolução. Sejam dois inteiros a e b na base B, então

$$a = a_0 + a_1 B + \ldots + a_{m-1} B^{m-1} \quad e$$

$$b = b_0 + b_1 B + \ldots + b_{m-1} B^{m-1}, \quad \text{onde } a_j \text{ eb } j \text{ são}$$

inteiros tais que $-1 < a_j, b_j < B$. Se escrevemos $a=(a_k)$, $b=(b_k)$, então o produto c=a.b é igual a

$$c = \sum_{i=0}^{2m-2} c_i B^i, \text{ onde } c_i' \text{ é dado por}$$

$$c_i = \sum_{j=0}^{i} a_j b_{i-j} = a*b \ .$$

Se n for muito grande, a convolução torna-se uma operação computacionalmente cara, ou seja, requer muitas multiplicações e adições.Os capítulos seguintes mostrarão que a transformada discreta de Fourier é uma operação mais barata, por isso vamos encontrar uma fórmula para a convolução em termos da transformada.

Sejam $X = (X_k)$, $Z = (Z_k)$ e $U = (U_k) = X * Z$. Se denotamos por $\hat{X}$, $\hat{Z}$ e $\hat{U}$ as transformadas de X, Z e U, respectivamente, então temos

$$\hat{U}_k = \sum_{j=0}^{n-1} U_i W^{-jk} = \sum_{j=0}^{n-1} \left(\sum_{m=0}^{n-1} X_m Z_{j-m} \right) W^{-jk} = \sum_{m=0}^{n-1} X_m \sum_{j=0}^{n-1} Z_{j-m} W^{-jk} \ .$$

Devido a periodicidade, podemos trocar a soma com respeito a j por p=j-m, e temos

$$\hat{U}_k = \sum_{m=0}^{n-1} X_m W^{-mk} \sum_{p=0}^{n-1} Z_p W^{-pk}, \text{ obtendo, portanto}$$

$$(9) \qquad \hat{U} = \hat{X}.\hat{Z}.$$

Equivalentemente na forma matricial, temos

$$F\,U = F\,X.F\,Z, \text{ de modo que vale}$$

(10) $U = F^{*}(F X . F Z)$

Do desenvolvimento acima, fica demonstrado o seguinte

Teorma da covolução: A convolução de duas seqüências periódicas X e Z de período n é dada por (10).

CAPÍTULO II

1.A TRANSFORMADA RÁPIDA DE FOURIER

A transformada rápida de Fourier tem, aproximadamente, 20 anos(Cooley e Tukey-1965) e surgiu da necessidade de calcular mais rapidamente a transformada discreta de Fourier.

Se seguimos um algoritmo natural para calcularmos os valores

$$(1) \quad A_r = \sum_{k=0}^{N-1} X_k e^{-2\pi irk/N} \qquad r=0,1,\ldots,n-1,$$

teríamos, para cada r, N multiplicaçõoes complexas e N adições, resultando um total de N^2 multiplicações e adições. Supondo o tempo de computação proporcional ao número de operações envolvidas, segue que o tempo para computar a transformada discreta de Fourier X_k, k=0,1,...,N-1, é da ordem de N^2 e escrevemos

$$(2) \qquad O(N^2)$$

A transformada rápida de Fourier (FFT) é um algoritmo que reduz o número de operações envolvidas na transformada discreta de Fourier, reduzindo, conseqüentemente, o tempo de computação.

Vamos supor, inicialmente que N, o tamanho da amostra, seja par. Dividimos então a seqüência $X_0,\ldots,X_{N-1}$ em duas seqüências Y_k e Z_k, cada uma com N/2 amostras. Especificadamente

$$(3)\qquad \begin{matrix} Y_k = X_{2k} \\ Z_k = X_{2k-1} \end{matrix} \qquad k=0,1,\ldots,N/2-1.$$

Podemos calcular a transformada discreta de Fourier das seqüências Y_k e Z_k, definidas por

$$(4)\qquad \begin{matrix} B_r = \sum_{k=0}^{N/2-1} Y_r\, e^{-4\pi irk/N} \\ C_r = \sum_{k=0}^{N/2-1} Z_k\, e^{-4\pi irk/N} \end{matrix} \qquad ,r=0,\ldots,N/2-1.$$

A_r, a transformada que queremos, pode ser calculada por

$$A_r = \sum_{k=0}^{N/2-1} \left(Y_k\, e^{-4\pi irk/N} + Z_k\, e^{-4\pi ir(2k+1)/N}\right).$$

Desenvolvendo, temos

$$A_r = \sum_{k=0}^{N/2-1} Y_k\, e^{-4\pi irk/N} + e^{-2\pi ir/N} \sum_{k=0}^{N/2-1} Z_k\, e^{-4\pi irk/N}$$

ou seja,

$$(5)\qquad A_r = B_r + e^{-2\pi ir/N} C_r,\ r=0,\ldots,N/2-1.$$

Do fato que

$$B_r = B_{r+N/2} = B_{2r+N/2} = \ldots \text{ e}$$

$$C_r = C_{r+N/2} = C_{2r+N/2} = \ldots \text{, temos que}$$

$$A_{r+N/2} = B_r + e^{-2\pi i(r+N/2)/N} C_r$$

$$= B_r - e^{-2\pi ir/N}.$$

Assim, A_r pode ser calculado por

$$A_r = B_r + W^r C_r, \quad 0 \leq r \leq N/2-1$$

(6)

$$A_{r+N/2} = B_r - W^r C_r, \quad 0 \leq r \leq N/2-1, \text{com } W = e^{-2\pi i/N}.$$

Obtivemos, portanto, um método que nos permite calcular transformada discreta de Fourier de N pontos através do cálculo de duas transformadas de N/2 pontos. O tempo de computação das transformadas B_r e C_r das seqüências Y_k e Z_k é $O((N/2)^2)$. Usando as equações (7), segue que o tempo de computação da tranformada A_r é $2.O((N/2)^2)$ mais N operações (N multiplicações e N adições).

Vamos ilustrar com um gráfico, usando N=8. No gráfico, cada nó representa uma variável. As setas indicam a contribuição aditiva da variável de onde partem para a variável onde chegam. Se o peso da contribuição não é a unidade, está indicado o valor na ponta da seta.

Podemos aplicar agora o mesmo processo para o cálculo da transformada discreta de N/2 pontos e obtermos assim o cálculo da transformada de N amostras através de quatro transformadas de N/4 amostras.

Essas reduções podem ser feitas até que o número de amostras resultante em cada seqüência seja par. Assim, se $N=2^m$, podemos fazer m reduções aplicando as equações (4) a (7) a N, N/2,..., até 2. A transformada discreta de Fourier de uma seqüência de um ponto é, naturalmente, o próprio ponto.

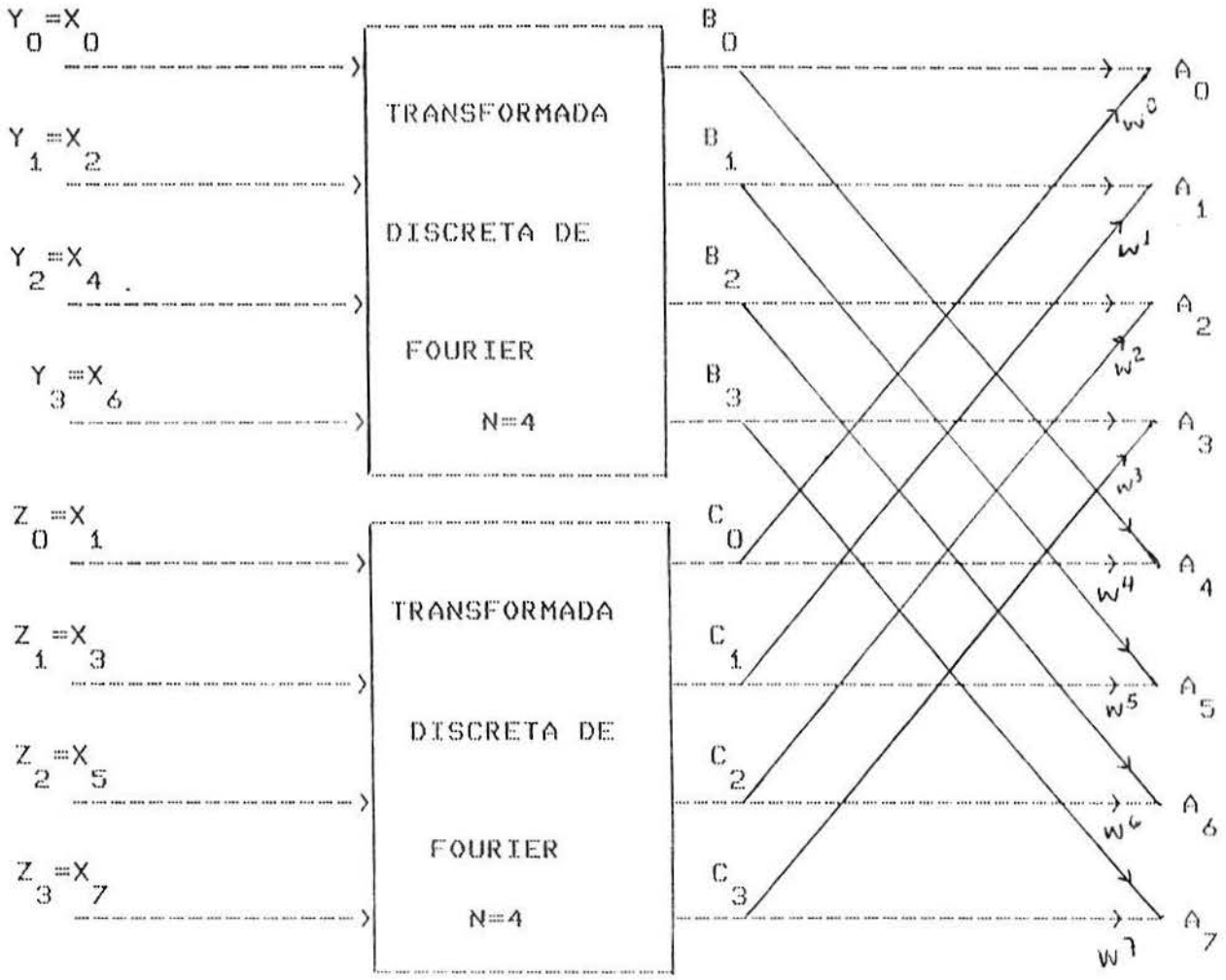

Fig. 1

Observe-se que $W^{n} = -W^{n-N/2}$. Usando este fato juntamente com a equação (7) para o cálculo de A_r, não serão mais necessárias N multiplicações e sim N/2 multiplicações complexas, embora o número de adições não se altere. Portanto, o tempo de computação é $2.0((N/2)^{2})+N/2$, se pensarmos somente nas multiplicações.

As reduções sucessivas para N=8 são continuadas nos gráficos das fig. 2 e fig. 3.

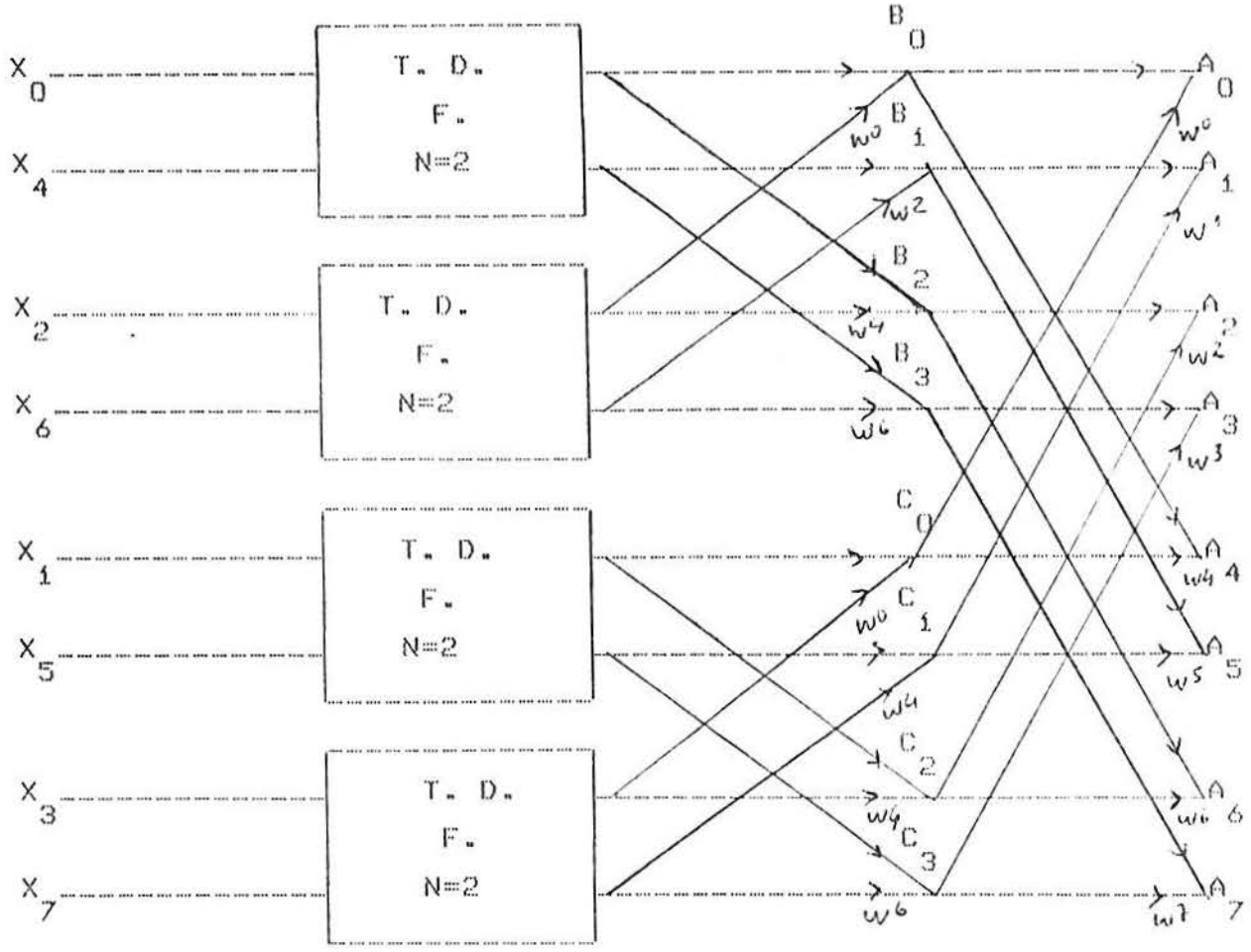
X0
X4
X2
X6
X1
X5
X3
X7
T. D. F. N=2
T. D. F. N=2
T. D. F. N=2
T. D. F. N=2
B0
B1
B2
B3
C0
C1
C2
C3
A0
A1
A2
A3
A4
A5
A6
A7
W0
W2
W4
W6
W1
W3
W5
W7

Fig. 2

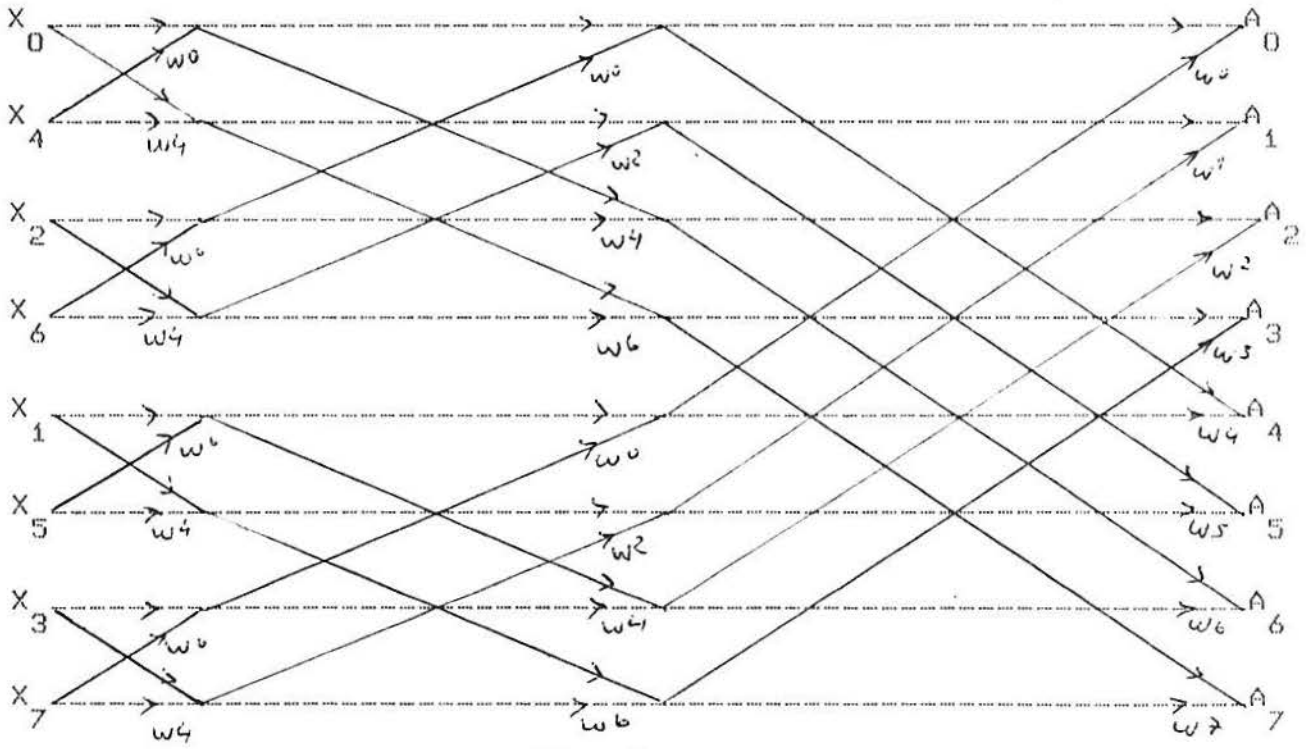
X0
X4
X2
X6
X1
X5
X3
X7
A0
A1
A2
A3
A4
A5
A6
A7
W0
W4
W2
W6
W1
W5
W3
W7

Fig. 3

Observemos que na fig. 3 a operação ficou reduzida somente a multiplicações e adições complexas. Note-se que a cada redução usamos as equações (7), o que significa que a cada redução temos N/2 multiplicações complexas e N adições. Assim, se $N=2^m$, podemos fazer m reduções, onde serão necessárias, no total m.N/2 multiplicações e m.N adições. Portanto, se $N=2^m$, o tempo de computação da tranformada discreta de Fourier da seqüência $X_0, X_1, \ldots, X_{N-1}$ é

$$(7) \qquad O(N.\log_2 N)$$

Em termos econômicos essa redução no tempo de computação é muito importante, tendo em vista que N, o tamanho da amostra é sempre muito grande nas aplicações práticas, tornando-se N^2 significativamente maior do que $N.\log_2 N$.

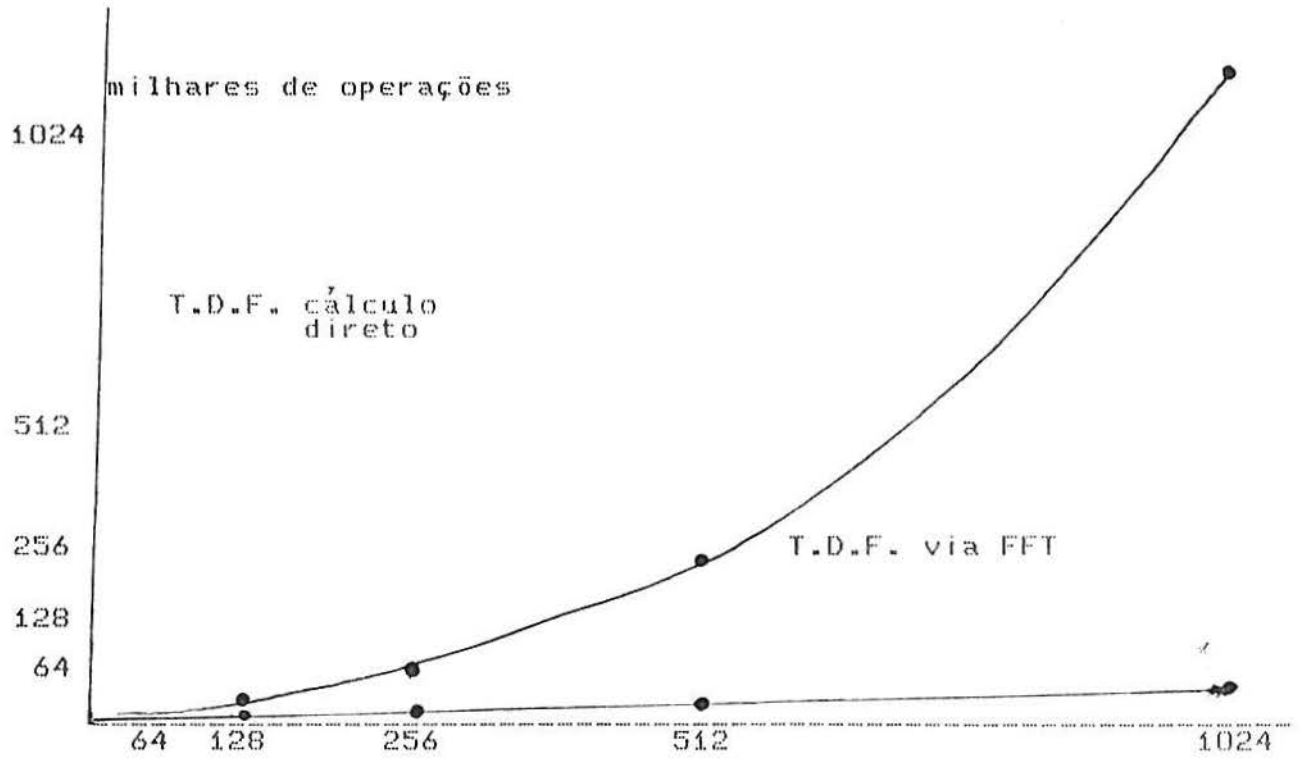

Fig. 4

A figura 4 mostra a relação entre o número de dados e o número de operações requeridas pela transformada discreta de de Fourier via FFT e por cálculo direto.

Se N não é uma potencia de dois, mas tem um fator primo p, as equações (3) a (6) tem um desenvolvimento análogo para p diferentes seqüências $Y_k^{(j)}=X_{pk+j}$, cada uma com N/p amostras e transformada de Fourier $B_r^{(j)}$. A transformada da seqüência $X_0,\ldots,X_{N-1}$ pode ser computada usando as p simples transformadas, isto é,

$$(8)\quad A_{r+m(N/p)}=\sum_{j=0}^{p-1} B_r^{(j)}\, W^{j(r+m(N/p))}, \qquad W=e^{-2\pi i/N},$$

com

$$(9)\quad \begin{matrix} m=0.1,\ldots,p-1 \\ r=0,1,\ldots,N/p-1 \end{matrix}$$

O cálculo das transformadas pode ser reduzido ainda mais se N tem outros fatores primos.

Outro problema de interesse prático é o cálculo da transformada inversa de Fourier. Dada a seqüência $X_0,\ldots,X_{N-1}$, com transformada discreta de Fourier A_r, podemos obter X_l a partir de A_r através de

$$(10)\quad X_l=1/N\sum_{r=0}^{N-1} A_r W^{-rl}, \quad l=0,1,\ldots,N-1, \quad W=e^{-2\pi i/N}$$

Admitindo a validade da equação (11), verifica-se que o cálculo da transformada inversa pode ser feito rapidamente usando a mesma

descrição acima , trocando X_k por A_r, tomando o cuidado de multiplicar pelo fator 1/N e mudar o sinal do expoente de W. Para mostrarmos a validade da equação (11), utilizamos a seguinte

Proposição 1: Se $W=e^{-2\pi i/N}$, então, para k inteiro temos

$$(11) \qquad \sum_{j=0}^{N-1} W^{jk} = \begin{cases} N, & \text{se } k=0 \pmod N \\ 0, & \text{c.c.} \end{cases}$$

Prova: Como $W^N=1$ e $k=0 \pmod N$, isto é, $k=lN$, temos $W^k=(W^N)^l=1$.

Portanto $\sum_{j=0}^{N-1} W^{jk}=N$.

Se $k=0 \pmod N$, vale que

$$(1-W^k)(1+W^k+\ldots+W^{k(N-1)})=1-W^{kN}$$

isto é,

$$(1-W^k)\sum_{j=0}^{N-1} W^{jk}=1-W^{kN}=0.$$

Como $1-W^k=0$, segue o resultado.#

Mostraremos agora que a equação (1) implica a equação (10). de maneira análoga mostra-se a recíproca. Inserindo a equação (1) na equação (10), temos

$$(12) \qquad \sum_{r=0}^{N-1} \sum_{k=0}^{N-1} (X_k/N)\, W^{r(k-1)}$$

Invertendo a ordem dos somatórios, vale, em (13)

$$(13) \qquad \sum_{k=0}^{N-1} (X_k/N) \sum_{r=0}^{N-1} Wr(k-1).$$

Da proposição 1, segue

$$(14) \qquad \sum_{r=0}^{N-1} W^{r(k-1)} = \begin{cases} N, & \text{se } k=1 \\ 0, & \text{c.c.} \end{cases}$$

Assim, substituindo (15) em (14), mostramos que

$$(1/N) \sum_{r=0}^{N-1} A_r W^{r1} = X_1 .$$

CAPÍTULO III

A FFT EM OPERAÇÕES COM POLINOMIOS

1.INTRODUÇÃO:

Seja F um corpo e suponhamos que queremos calcular, em F(x), uma expressão $h(x)=e(a_1(x),\ldots,a_s(x))$. Por exemplo: $e(a(x),b(x))=a(x).b(x)$. Em geral essa expressão é difícil de ser calculada em F(x), principalmente se o grau dos polinômios envolvidos é elevado. A estratégia a ser usada é passar da aritmética difícil de F(x) para a aritmética mais simples de F, utilizando imagems homomórficas.

Com esse objetivo em mente, introduzimos um esquema de avaliação-interpolação, que é um método para avaliar a expressão $h(x)=e(a_1(x),\ldots,a_s(x))$, onde $a_1(x),\ldots,a_s(x)$ são polinômios de F(x), abreviados por a(x). Seja n uma limitação para o grau de h(x) e sejam b_k $(k=0,\ldots,n-1)$ n pontos distintos escolhidos de F. Anotamos por $a_i^{(k)}=a_i(b_k)$ $(k=0,1,\ldots,n-1)$ e procedemos do seguinte modo

Esquema Avaliação-Interpolação

Passo 1:Para $k=0,1,\ldots,n-1$

(i) computar $a_i^{(k)}=a_i(b_k)$ $(i=0,\ldots,s)$

(ii) computar $c_k=e(a^{(k)})=e(a_1^{(k)},\ldots,a_s^{(k)})$

Passo 2:Encontrar o polinômio U(x) de grau menor do que n tal que

$$U(b_k)=c_k \quad (k=0,1,\ldots,n-1)$$

Passo 3: Fazer h(x)=U(x).#

O esquema pode ser representado pela figura 1

Fig. 1

Teorema 1:O algoritmo computa U(x)=h(x), onde h(x)=e(a(x))

Prova: $H_{bk}: a(x) \longrightarrow a(b_k)=a^{(k)}$ é homomorfismo de F(x) sobre F.

Assim,

$H_{bk}(\ e(a(x))\ )=e(H_{bk})$, o que implica dizer que c_k computado no passo 1 (ii) satisfaz

$$h(b_k)=c_k .$$

Como h(X) e U(x), computado no passo 2, são dois polinômios de grau menor do que n que interpolam n pontos distintos (b_k,c_k) de F^2 (k=0,1,...,n-1), temos, pela unicidade da interpolação, que U(x)=h(x).#

A questão que se coloca aqui é a seguinte: o esquema avaliação-interpolação é mais econômico que o algoritmo usual em F(x) ?

Vejamos no caso específico de h(x)=e(a(x),b(x))=a(x).b(x), a multiplicação de dois polinômios.

Lembremos que o algoritmo clássico para multiplicação de dois polinômios de graus m e n, respectivamente, requer m.n multiplicações, isto é, o tempo de computação é O(m.n).Consideraremos o caso de dois polinômios de grau n, o que permite concluir que o tempo requerido pelo algoritmo normal é $O(n^2)$.

O esquema avaliação-interpolação, para a(x).b(x), requer:

Passo 1 (i): Realiza a avaliação de dois polinômios de grau n em N=2n+1 pontos (pois o grau de h(x) < 2n). Usando a regra de Horner[1], que requer n multiplicações e n adições para cada avaliação, teremos um total de 2((2n+1).n) multiplicações e adições. O tempo requerido e', portanto, de $O(n^2)$.

passo 1 (ii):Efetua N=2n+1 multiplicações sobre F; O(n) e o tempo requerido.

Passo 2:Realiza uma interpolação polinomial de N=2n+1 pontos, requerendo um tempo de $O(n^2)$, usando o método de interpolação de Newton.

Portanto, o novo método não apresenta, em princípio, vantagem sobre o algoritmo clássico. Vamos usar a transformada rápida de Fourier para reduzir o número de multiplicações no passo 1 (i) do algoritmo, na avaliação, e também no passo 2, a interpolação.

[1] A regra de Horner é uma regra ótima para a avaliação de um polinômio em um ponto, isto é , não podemos avaliar um polinômio de grau n em um ponto em menos do que n multiplicações e n adições. A regra simplesmente escreve $a(x)=a_0+a_1x+\ldots+a_nx^n$ como $a(x)=((\ldots((a_nx+a_{n-1})x+a_{n-2})\ldots+a_2)x+a_1)x+a_0$

2.AVALIAÇÃO MÚLTIPLA RÁPIDA

Estamos interessados em avaliar o polinômio $a(x)=a_0+a_1x+a_2+\ldots+a_{N-1}x^{N-1}$ de F(x) em N pontos $x_0,\ldots,x_{N-1}$ do corpo F. Podemos resolver esse problema em $O(N^2)$ operações, usando a regra de Horner. Assumindo que $N=2^m$ e que F possui uma N-raiz primitiva da unidade w, podemos reduzir o número de operações para $O(N.\log_2 N)$, se $x_i=w^i$, $i=0,1,\ldots,N-1$.

Vamos decompor a(x):

$$a(x)=a_0+a_1x+a_2x^2+\ldots+a_{N-1}x^{N-1}$$

$$=(a_0+a_2x^2+\ldots+a_{N-2}x^{N-2})+(a_1x+\ldots+a_{N-1}x^{N-1}).$$

Se $y=x^2$,

$$a(x)=(a_0+a_2y+\ldots+a_{N-2}y^{(N-2)/2})+x(a_1+\ldots+a_{N-1}x^{(N-2)/2}$$

$=b(y)+x\,c(y)$, onde grau de b e grau de c é $(N-2)/2$.

Feita essa decomposição, o problema original de avaliar o polinômio a(x) em $x_i=w^i$ $(i=0,\ldots,N-1)$, se reduz a avaliar os polinômios b(x) e c(x) em $y_0,\ldots,y_{N-1}$, com $y_i=x_i^2$. Como $w^{k+N/2}=-w^k$, logo $y_i=y_{i+N/2}$, o que implica, de fato, na avaliação de somente N/2 pontos dinstintos $y_0,y_1,\ldots,y_{N/2-1}$. Podemos achar o valor de $a(x_i)$ através do cálculo

$$a(x_i)=b(y_i)+x_i\,c(y_i), \qquad i=0,\ldots,N/2$$

(1)

$$a(x_i)=b(y_i)-x_i\,c(y_i), \qquad i=N/2+1,\ldots,N-1.$$

2.AVALIAÇÃO MÚLTIPLA RÁPIDA

Estamos interessados em avaliar o polinômio $a(x)=a_0+a_1x+a_2+\ldots+a_{N-1}x^{N-1}$ de F(x) em N pontos $x_0,\ldots,x_{N-1}$ do corpo F. Podemos resolver esse problema em $O(N^2)$ operações, usando a regra de Horner. Assumindo que $N=2^m$ e que F possui uma N-raiz primitiva da unidade w, podemos reduzir o número de operações para $O(N.\log_2 N)$, se $x_i=w^i$, $i=0,1,\ldots,N-1$.

Vamos decompor a(x):

$$a(x)=a_0+a_1x+a_2x^2+\ldots+a_{N-1}x^{N-1}$$

$$=(a_0+a_2x^2+\ldots+a_{N-2}x^{N-2})+(a_1x+\ldots+a_{N-1}x^{N-1}).$$

Se $y=x^2$,

$$a(x)=(a_0+a_2y+\ldots+a_{N-2}y^{(N-2)/2})+x(a_1+\ldots+a_{N-1}x^{(N-2)/2}$$

$$=b(y)+x\,c(y),$$ onde grau de b e grau de c é (N-2)/2.

Feita essa decomposição, o problema original de avaliar o polinômio a(x) em $x_i=w^i$ $(i=0,\ldots,N-1)$, se reduz a avaliar os polinômios b(x) e c(x) em $y_0,\ldots,y_{N-1}$, com $y_i=x_i^2$. Como $w^{k+N/2}=-w^k$, logo $y_i=y_{i+N/2}$, o que implica, de fato, na avaliação de somente N/2 pontos dinstintos $y_0,y_1,\ldots,y_{N/2-1}$. Podemos achar o valor de $a(x_i)$ através do cálculo

$$a(x_i)=b(y_i)+x_i\,c(y_i), \qquad i=0,\ldots,N/2$$

(1)

$$a(x_i)=b(y_i)-x_i\,c(y_i), \qquad i=N/2+1,\ldots,N-1.$$

Portanto, o problema de tamanho N se reduziu a dois subproblemas de tamanho N/2: avaliar dois polinômios de grau (N-2)/2 em N/2 pontos, acrescentando-se mais N/2 multiplicações realizadas em (1).

O processo pode ser aplicado recursivamente aos polinômios b(x) , c(x) e suas decomposições porque começamos com w uma N-raiz primitiva da unidade, então w^2 é uma N/2 raiz primitiva da unidade, w^4 é uma N/4-raiz primitiva da unidade, etc. Olhemos para M(N), o número de multiplicações, que é dado por duas parcelas :

-2.M(N/2), devido a redução do problema

-N/2, devido ao uso de (1).

Assim, sendo $N=2^m$, temos:

$M(2^m)=2.M(N/2)+2^{m-1}$, repetindo o processo,

$M(2^m)=2.(2.M(2^{m-2})+2^{m-1}+2^{m-1}$, até chegarmos ao final do processo com m aplicações, isto é

$M(2^m)=2^m.M(1)+m.2^{m-1}$ multiplicações. Como M(1)=0 (é o número de multiplicações para avaliar um polinômio de grau zero em um ponto), temos que

$$(2) \qquad M(N)=m.2^{m-1}=N/2.\log_2 N.$$

Reduzimos, conforme planejamos, o número de operações, reduzindo o tempo de computação para a multiavaliação. Temos que o tempo de computação requerido para avaliar um

polinômio de grau N-1 em N pontos é $O(N.\log_2 N)$.

O processo pode ser resumido no seguinte

Algoritmo 1:

INPUT:-Polinômio de F(x) $a(x)=a_0+\ldots+a_{N-1}x^{N-1}$, com $N=2^m$

-w uma N-raiz primitiva da unidade em F

OUTPUT:-O vetor $A=(A_0,\ldots,A_{N-1})$, onde $A_k=a(w^k)$

MÉTODO:O seguinte procedimento de alto nível em subrotina de linguagem PASCAL

```
PROCEDURE FFT(N, a(x), W, A)
IF N=1 THEN
        A0:=a0
    ELSE
    BEGIN
        n:=N/2;
        b(x):=a0+a2x^2+....+an-2x^(n-2);
        c(x):=a1x+a3x^3+....+an-1x^(n-1);
        FFT(n, b(x), W^2, B);
        FFT(n, c(x), W^2, C);
        FOR K:=0 TO n-1 DO
            BEGIN
                AK:=BK+W^K cK;
                AK+n:=BK-W^K CK
            END;
END.
```

3.INTERPOLAÇÃO RÁPIDA:

Com relação a interpolação, temos o seguinte problema associado: Dada w uma N-raiz primitiva da unidade em F e $N=2^m$, queremos encontrar a_i $(i=0,\ldots,N-1)$ tal que o polinômio $a(x)=a_0+a_1x+\ldots+a_{N-1}x^{N-1}$ satisfaça $a(w^k)=b_k$ $(k=0,\ldots,N-1)$, para b_k dados.

Esse problema pode ser examinado pela matriz de Vandermonde

$$V(1,w,\ldots,w^{N-1})=\begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & w & w^2 & \ldots & w^{N-1} \\ 1 & w^2 & w^4 & \ldots & w^{2(N-1)} \\ \vdots & & & & \\ 1 & w^{N-1} & w^{2(N-1)} & \ldots & w^{(N-1)^2} \end{bmatrix}$$

Sejam $a=(a_0,\ldots,a_{N-1})$ e $b=(b_0,\ldots,b_{N-1})$. Temos, para $a(x)=a_0+\ldots+a_{N-1}x^{N-1}$ e $V=V(1,\ldots,w^{N-1})$, que

$$Va=b \iff b_k=a(w_k).$$

Como V é inversível, temos garantida a existência única dos a_i, isto é,

$$Va=b \iff a=V^{-1}b \iff b_k=a(w^k).$$

Isto significa que o problema da multiavaliação pode ser pensada como uma multiplicação da forma Va, com V matriz de Vandermonde e a FFT é um algoritmo que serve para computar

rapidamente Va. Também a interpolação é um produto de matrizes $V^{-1}b$, com $V=V(1,\ldots,w^{N-1})$. Veremos que V^{-1} pode ser calculada facilmente a partir de V pela

Proposição 1: Seja w uma N-raiz primitiva da unidade no corpo F, no qual exista 1/N. Então

$$V^{-1}(1,\ldots,w^{N-1})=1/N\ V(1,w^{-1},\ldots,w^{-(N-1)})$$

Prova: Desde que w^{-1} é uma N-raiz primitiva da unidade, mostraremos que

$$K=(k_{ij})=V(1,w,\ldots,w^{N-1}).V(1,\ldots,w^{-N+1})=\begin{bmatrix} N & 0 & \ldots & 0 \\ 0 & N & \ldots & 0 \\ \vdots & & & \\ 0 & 0 & \ldots & N \end{bmatrix}$$

Ora,

$$k_{ij}=\sum_{s=0}^{N-1} w^{is}\ w^{-js}=\sum_{s=0}^{N-1} w^{s(i-j)},$$

se i=j, temos que $w^{s(i-j)}=1$, então $k_{ij}=N$. Se i≠j, usamos que, para x≠1, vale

$1+x+\ldots+x^{N-1}=(x^N-1)/(x-1)$ e, dessa forma, sendo $0<|i-j|<N$, segue que

$$\sum_{s=0}^{N-1} w^{s(i-j)} = ((w^{i-j})^N - 1)/(w^{i-j}-1)=0.$$

Resolvemos, portanto, o problema da interpolação usando o mesmo algoritmo da multiavaliação. Mais objetivamente, temos o

Algoritmo 2:

INPUT:- O vetor $b=(b_0,\ldots,b_{N-1})$ de F^N
- w uma N-raiz primitiva da unidade de F

OUTPUT:-$a(x)=a_0+\ldots+a_{N-1}x^{N-1}$, com $a(w^k)=b_k$ $(k=0,\ldots,N-1)$

MÉTODO:O seguinte procedimento de alto nível em linguagem PASCAL.

```
PROCEDURE FFI(N, b, W, a(x));

        BEGIN
              b(x):=Σ(i=0 até N-1) b_i x^i;

              FFT(N, b(x), W^-1, C);
              a(x):=Σ(i=0 até N-1) (1/NC_i)X^i;
END.
```

Claramente o algoritmo acima opera no mesmo tempo - $O(N.\log_2 N$- que a FFT , mostrando que a interpolação tambem pode ser implementada rapidamente.

Por último queremos observar que o método para operações com polinômios apresentado tem alguns inconvinientes. O principal é que o corpo F em questão deve ter uma N-raiz primitiva da unidade. O corpo dos complexos sempre tem N-raiz primitiva da unidade: $w=e^{2\pi i/N}$. Nesse caso o inconveniente e' as operações , já que as multiplicações envolvidas são de números complexos. Se os polinômios envolvidos são inteiros (ou racinais)

é possível realizar as operações sobre um corpo finito Zp. A viabilidade desse método é o objetivo do próximo capítulo.

CAPÍTULO IV

PRATICABILIDADE DA FFT MÓDULO p

Para que possamos aplicar eficientemente o algoritmo FFT em corpos finitos Zp, com p um primo, é preciso responder basicamente duas questões:

(i) Zp tem N-raiz primitiva da unidade ?

(ii) Dado Zp com uma N-raiz primitiva da unidade, podemos encontrá-la eficientemente ?

Ao final deste capítulo teremos respondido ambas as quetões com relativo sucesso.

Se $N=2^m$, estamos buscando primos p para os quais Zp tenha uma N-raiz primitiva da unidade (nas aplicações p é sempre muito grande, na ordem de 10^9). Invocamos o seguinte

Teorema 1: Zp tem N-raiz primitiva da unidade se e somente se

N divide p-1

Prova: Seja N tal que N divide p-1. Como $(Zp^*,.)$ é um grupo cíclico , existe um elemento a de Zp tal que a é primitivo, isto é, $a^{p-1}=1$ e $a^n \neq 1$ se $1< n <p-1$.

Portanto, $b=a^{(p-1)/N}$ é N-raiz primitiva da unidade.

A outra direção segue do teorema de Lagrange (a ordem de um elemento divide a ordem do grupo).

Assim, para computar FFT módulo p de tamanho $N=2^m$, precisamos de números primos p tal que $2^m/p-1$, ou seja, primos da forma $p=2^r k+1$ ($r > m+1$, k ímpar).

O seguinte resultado da teoria analítica de números garante a existência em abundância de primos p da forma $p=2^r k+1$:

O número de primos p da forma $p=2^r k+1 < x$ é, aproximadamente

$$(x/\log x)/2^{r-1}.$$

Por exemplo se a palavra de um computador é de tamanho 2^{31}, o número de primos p da forma $p=2^r k+1$ (k ímpar), com $r > 20$, menores do que $x=2^{31}$ é, aproximadamente

$$2^{31}/\log 2^{31}.2^{19}=2^{12}/31.\log 2\sim 180.$$

Tais 180 primos podem ser usados para computar FFT de tamanho até $2^{20}\sim 10^6$.

Para respondermos a segunda questão, que trata de achar um meio eficiente de encontrar uma raiz primitiva do corpo Zp, ela pode ser reduzida a encontrar um elemento primitivo de Zp, pois se a é primitivo, então $b=a^{(p-1)/N}$ é uma N-raiz primitiva da unidade.

Como encontrar um elemento primitivo em Zp, com $p=2^r k+1$? Sabemos que o número de elementos primitivos em Zp é

dado por H(p-1), H função de Euler. A teoria analítica de números nos diz que o valor médio de H(n) é $6n/\pi^2$, para n inteiro.

Se $p-1=2^r k$ (k ímpar), escrevemos

$$H(p-1)=H(2^r).H(k)=2^{r-1}H(k).$$

Agora, o valor médio de H(k), para k ímpar, deve ser maior do que $6k/\pi^2$, já que seu valor médio sobre k par é $< k/2 < 6k/\pi^2$.

Assim, $H(p-1) > 2^{r-1} 6k/\pi^2 = 3.2^r k/\pi^2 = (p-1)3/\pi^2$, significando que o número de elementos primitivos em Zp é, em média, maior do que $3(p-1)/\pi^2$. Em termos probabilísticos, se um elemento é retirado ao acaso de Zp, ele tem a probabilidade $3/\pi^2 \sim 0,3$ de ser primitivo.

Portanto, para acharmos um elemento primitivo em Zp, testamos 2,3,... . A probabilidade acima nos deixa seguros de que não será preciso testar muitos elementos até alcançarmos o objetivo.

Para testarmos os elementos seria absurdo verificar se $a^n \neq 1$, se $1 < n < p-1$. Felizmente, temos um resultado algébrico completo e elegante que vai facilitar a verificação.

Teorema 2: a um elemento de Zp é primitivo se e somente se

$$a^{(p-1)/q} \neq 1, \text{ para todo q fator primo de } p-1.$$

Prova: Se a é primitivo, $a^n \neq 1$, para $1 < n < p-1$.

Se a tem ordem $n < p-1$, então $kn=p-1$. Seja $k=qr$, onde q é fator primo. Então,

$a^{(p-1)/q} = a^{rn} = (a^{n})^{r} = 1$, o que

mostra a outra direção do teorema.#

Exemplo 1:Vamos encontrar um elemento primitivo em Zp, com p=13.

Sendo $p-1 = 12 = 2^{2}.3$, a e' primitivo se e somente se a^{4}, $a^{6} \neq 1$.

Temos:.

$$2^{4} = 16 = 3 \pmod{13}$$

$$2^{6} = 2^{4}.2^{2} = 3.4 = 12 \pmod{13}.$$

Assim, 2 e' elemento primitivo de Zp, com p=13.

Resumindo, temos o seguinte

Algoritmo 1:

INPUT:-Primo $p=2^{r}k+1$, com $r > m$, onde $N=2^{m}$ e' o tamanho da FFT a ser computada.

OUTPUT:-Um elemento primitivo de Zp

Passo 1:Determinar a fatorização prima de p-1

Passo 2:Testar a=2,3,... , usando o teorma 2.

No capítulo VI seguinte apresentamos um programa, em linguagem PASCAL, que encontra o menor elemento primitivo positivo de Zp, com p um primo < do que 32500.

CAPÍTULO V

MULTIPLICAÇÃO DE INTEIROS

1.INTRODUÇÃO:

. Por vários anos muitas pessoas usavam a estreita relação que existe entre a multiplicação de polinômios e a multiplicação de inteiros (Pollard), (Aho). Se $a=(a_{n-1} \ldots a_0)_B$ é um inteiro de n dígitos na base B, então a representa o valor do polinômio associado $a(x)=a_0+\ldots+a_{n-1}x^{n-1}$ em x=B, isto é, a=a(B). Mas somente em 1971 que Shönhage e Strassen demonstraram uma maneira pela qual a analogia pode ser explorada, de modo que a multiplicação de dois inteiros de n dígitos na base B pode ser implementada em O(n.log n.log log n) (Aho). Trataremos neste capítulo de um algoritmo sugerido por Pollard (1971) e analisado por Lipson (1974) que multiplica dois inteiros de n dígitos em O(n.log n) para n < N, onde N é "bastante grande", porém limitado (Borodin).

Estamos interessados, evidentemente, em multiplicar inteiros muito grandes, daí escolhemos como base B a maior potência de dez que seja menor do que W, o tamanho da palavra do computador. Por exemplo, se o computador tem 16 bits, então $W=2^{15}$ (mais o sinal) e a base escolhida deve ser B=10.000. Assim o número 71348946235 nessa base é 0713.4894.6235, apresentando somente 3 dígitos. Observe-se que essa base B facilita a passagem da base dez para B e reciprocamente. Ainda temos como vantagem ao escolher tal base B o fato de que os coeficientes do polinômio

associado ao inteiro a são todos de precisão simples, ou seja, tem somente "um" dígito.

O tamanho da base B nos dá uma idéia da dificuldade que teríamos em calcular a FFT diretamente usando a ténica da multiplicação de polinômios dos capítulos precedentes.

O algoritmo a ser estudado utiliza um certo número de primos p_k, $B < p_k < W$ e o teorema chinês do resto para reduzir o problema em subproblemas menores.

2.0 PROBLEMA CHINES DO RESTO:

O problema chinês do resto (PCR) é um sistema de congruências lineares

$$(1) \qquad u = r_k \pmod{m_k} \qquad (k=0,\ldots,n-1)$$

sobre os inteiros.

Vamos nos restringir ao caso em que m_k, $k=0,\ldots,n-1$, são primos entre si dois a dois. Daremos um algoritmo que resolve tal problema.

Inicialmente tomemos o seguinte sistema de duas congruências

$$(2) \qquad u = r \pmod{m}$$

$$(3) \qquad u = s \pmod{n}$$

Onde u, r, s, m e n são inteiros com mdc(m,n)=1.

A solução de (2) é do tipo $u = r + t.m$, t inteiro. Encontraremos

um determinado t de modo que u também satisfaça (3).

Dado que mdc(m,n)=1, segue que existem x e y inteiros tais que $x.m + y.n = 1$, então $x.m - 1 = y.n$, ou seja, $x.m \equiv 1 \pmod{n}$. Assim, se tomamos

$t = (s-r).x \pmod{n}$, teremos

$u = (r + ((s-r).x \pmod{n}.m)) \pmod{n}$

$= (r + (s-r)) \pmod{n} = s \pmod{n}$.

Do desenvolvimento acima, segue o

Teorema 1: Sejam m e n inteiros relativamente primos, então o sistema

$u = r \pmod{m}$

$u = s \pmod{n}$ tem uma solução U inteira.

Tal solução pode ser obtida pelo

Algoritmo 1:

INPUT:-Um sistema de duas congruências inteiras

OUTPUT:-Uma solução U inteira

MÉTODO:-Subrotina abaixo, em linguagem PASCAL:

```
FUNCTION INVERSO(M,N);
BEGIN
    I:=0;
    REPEAT
        I:=I+1
    UNTIL (I*M) MOD N = 1;
    INVERSO:=I;
END;
```

```
PROCEDURE SOLUÇÃO(M,N);
BEGIN
    X:=INVERSO(M,N);
    T:=((S-R) * X) MOD N;
    U:=R + T*M
END;  .
```

Consideremos agora as n congruências

$$u = r_0 \pmod{m_0}$$
$$\vdots$$
$$u = r_{n-1} \pmod{m_{n-1}}, \text{ com } mdc(m_i, m_j)=1, \text{ se } i=j.$$

Definimos

$$M_k = 1, \text{ se } k=0$$
$$M_k = m_0 . m_1 . \ldots . m_{k-1}, \text{ se } k>0$$

Lema 1: M_k e m_k são relativamente primos, $0 < k < n$.

Prova: Se k=0, nada temos a mostrar, pois $M_k = 1$.

Se $k > 0$, supõe que existe $g > 1$ tal que g divide m_k e M_k.

Seja p fator primo de g. Como p divide M_k, então p divide m_j, para algum $0 < j < k$, assim $mdc(m_j, m_k) \geq p > 1$, o que é uma contradição.#

Para resolvermos (1), definimos a propridade p(k), k=0,1,...,n-1 de modo que após k iterações p(k) seja satisfeita, onde

$$(4) \quad p(k): \begin{cases} M=M_k \\ U = r_i \pmod{m_i} \quad (i=0,\ldots,k) \end{cases}$$

Após n-1 iterações U será a solução de (1) que buscamos.

Naturalmente que para a indução ter sucesso, definimos p(0) como sendo:

$$\begin{cases} M = 1 \\ U = r_0 \pmod{m_0}. \end{cases}$$

Por indução, podemos assumir que após k-1 iterações p(k-1) é válida :

$$M = M_{k-1}$$
$$U = r_i \pmod{m_i} \quad (i=0,\ldots,k-1).$$

Nosso objetivo é determinar U' e M' tal p(k) seja válida após k iterações.

Seja $M' = M_{k-1}.m_k = M_k$. Para definirmos U', lembremos que $mdc(M_k, m_k) = 1$, então, usamos o algoritmo 1 para calcular a solução U' do sistema

$$(5) \qquad u = U \pmod{M_k}$$

$$(6) \qquad u = r_k \pmod{m_k}$$

Vamos provar agora que U' assim definido satisfaz p(k):

Por (5), $U' = U \pmod{M_k}$, então $U' = U \pmod{m_i}$ $(i=0,\ldots,k-1)$.

Por hipótese de indução, temos que $U' = r_i \pmod{m_i}$ $(i=0,\ldots,k-1)$, que combinado com (6) completa a prova.#

Teorema 2:O PCR

$u = r_k \pmod{m_k}$ $(k=0,\ldots,n-1)$ tem uma solução inteira U que pode ser computada pelo

Algoritmo 2:

INPUT:-n congruências inteiras $u_k = r_k \pmod{m_k}$ $(k=0,\ldots,n-1)$

OUTPUT:-Uma solução inteira U

MÉTODO:-Subrotina abaixo, em linguagem PASCAL:

```
PROCEDURE N-SOLUÇÃO(N,U);

BEGIN

     M:=1;

     U:= R  MOD M ;
         0      0

     FOR K:=0 TO N-1 DO

     BEGIN

          M:=M*M   ;
                K-1

          X:=INVERSO(M,M  );
                      K

          T:=((R  -(U MOD M  ))*X) MOD M  ;
               K          K              K

          U:=U+T*M

     END;

END;
```

No cap,itulo VI apresentamos um programa completo que resolve o PCR.

3.SISTEMA DE CONGRUENCIAS POLINOMIAIS SOBRE Z(X):

Estamos interessados em encontrar uma solução U(x) em Z(x) da congruência

$$(7) \quad u(x) = p_k(x) \pmod{m_k} \quad (k=0,\ldots,n-1)$$

O lema seguinte reduz esse problema ao problema chinês do resto sobre os inteiros.

Lema 2: Sejam $a(x)=a_0+\ldots+a_n x^n$, $b(x)=b_0+\ldots+b_s x^s$ polinômios de $Z(x)$ e m um inteiro positivo. Então

$$a(x) = b(x) \pmod m \text{ se e somente se } a_i = b_i$$

prova: m divide $a(x)-b(x)$ se e somente se m divide $a_i - b_i$.#

Teorema 3: Sejam $p_k(x)= p_{0k}+p_{1k}x+\ldots+p_{qk}$, polinômios de $Z(x)$ e $m_k > 0$ $(k=0,\ldots,n-1)$, com m_k primos entre si dois a dois. Então o sistema de congruências polinomiais

$$u(x) = p_k(x) \pmod{m_k}$$

tem uma solução $U(x)$ pertencente a $Z(x)$.

Prova: Usando o lema 2, o j-ésimo coeficiente de qualquer solução polinomial de (7) é solução do PCR

$$u = p_{jk} \pmod{m_k}.\#$$

4.ALGORITMO DOS TRES PRIMOS: Apresentaremos agora o algoritmo de Pollard-Lipson referido acima

Algoritmo dos três primos

INPUT: $a=(a_{n-1}\ldots a_0)_B$ e $b=(b_{n-1}\ldots b_0)_B$ inteiros em base B

OUTPUT: c=ab

O algoritmo requer K primos da forma $p=2^e l+1 < W$, o tamanho da palavra do computador, com e suficientemente grande e K a ser determinado

Passo 1:

1.1 Para os K primos p_k ($B < p_k < W$):

Computar $c^{k}(x)=a(x)b(x)$ sobre $Zp(x)$ usando FFT.

1.2 Resolver o problema chinês do resto:

$$u(x) = c^{k}(x) \pmod{p_k} \qquad (k=0,1,\ldots,K-1)$$

1.3 $c(x)=U(x)$

Passo 2: $c=c(B)$.

Faz-se necessário determinar K, o número de primos a ser utilizado pelo algoritmo. Cada coeficiente $c_i = a_i b_0 + \ldots + a_0 b_i$ de c(x) é $< nB^2$. Então, no passo 1 temos que

$$(*) \quad p_0 p_1 \ldots p_{K-1} > nB^2.$$

Como cada $p_k > B$, segue que (*) é satisfeita se

$$B^K > nB^2, \text{ isto é, se}$$

$$K > \log_B n+2.$$

Assim, se $n \leq B$, três primos são suficientes.

Tendo escolhido K=3, temos uma restrição tácita em n, o tamanho do problema: $n \leq B$. Contudo, tendo em vista que nas aplicações B é sempre muito grande, tal restrição pode ser considerada irrelevante. Há, entretanto uma segunda imposição a n. O passo 1 do algoritmo requer três primos $p=2^e 1+1$, ($B \leq p < W$) com $2^e > 2n-1$ (grau de $c^{k}(x) + 1$). Então, sendo E o maior inteiro para o qual podemos encontrar três primos da forma requerida com expoente $e \geq E$, temos que $n \leq 2^{E-1}$.

Para analisarmos o tempo de computação necessária pelo algoritmo, vemos que o passo 1.1 requer $O(n.\log n)$

operações, enquanto o passo 1.2 requer O(n) operações (2n-1 problemas chineses do resto, cada um dos quais envolvendo três congruências inteiras.Pode-se mostrar que o passo 2, a avaliação polinomial requer O(n.log n) operações (Lipson). Portanto,podemos resumir os resultados obtidos no

Teorema 1: Seja um computador com palavra W. Então tal computador pode multiplicar dois inteiros de n dígitos na base B, $B < W$, em um tempo da O(n.log n), se

1. $n \leq B$
2. $n \leq 2^{E-1}$, onde E é o maior inteiro tal que podemos encontrar três primos $p=2^{e}1+1$ $(B \leq p < W)$ com $e \geq E$.

O nosso algoritmo reduz o tempo de computação para multiplicar dois inteiros de n dígitos de $O(n^2)$ para O(n.log n), mas é preciso lembrar que há restrições sobre n, o tamanho dos números a serem multiplicados. Vamos ver com um exemplo que o domínio de aplicação para esse algoritmo é extremamente grande.

Tomemos um computador com uma palavra de 32 bits, o que dá como $W=2^{31}-1$ e podemos escolher $B=10^{9}$. Portanto a primeira restrição deve ser entendida como $n \leq 10^{9}$.

Examinando a figura 1, podemos observar que existem três primos $B \leq p < W$ com expoente $e > 24$. Logo, a restriçõo $n \leq 2^{E-1}$, do teorema 1 significa

$$n < 2^{23} \sim 8{,}38 \times 10^{6}, \text{ ou seja, } E=24.$$

$p = 2^{e} l+1$ (l impar)	e	r=menor elemento primitivo de Zp
2013265921	27	31
2113929217	25	5
2130706433	24	3

Figura 1.

Temos, portanto, que um computador de palavra 32 bits pode multiplicar, usando o algoritmo 1, números inteiros que excedem a oito milhões de dígitos decimais.

CAPÍTULO VI

IMPLEMENTAÇÕES

Os programas a seguir apresentados são em linguagem PASCAL.

Programa MDC: Este programa calcula o MDC de dois inteiros positivos quaisquer menores do que 32500(restrição da máquina)

```
PROGRAM MDC_2;
VAR
   YV:ARRAY[0..4] OF INTEGER;
   A,B:INTEGER; K:REAL;
BEGIN
     WRITELN('ESTE PROGRAMA CALCULA O MDC DE DOIS INTEIROS 'A' E 'B'');
     WRITELN('');
     WRITELN('ESCOLHA DOIS INTEIRO MENORES QUE 32500');
     WRITELN('A ?');READLN(A);WRITELN('B ?');READLN(B);
     YV[0]:=A;
     YV[1]:=B;
     WHILE YV[1]<>0 DO BEGIN
                           YV[2]:=YV[1];
                           YV[1]:=YV[0]-((YV[0] DIV YV[1]))*YV[1];
                           YV[0]:=YV[2];END;
     WRITELN('O MDC DE A = ',A,'    E    B = ',B,'   E   ',YV[0]);
END.
```

Programa primos: Este programa lista e conta todos os primos menores do que n, um dado inteiro menor do que 32500.

```
PROGRAM PRIMOS_N;
VAR
  Y:ARRAY[0..32500] OF INTEGER;
  I,N,M:INTEGER;
BEGIN
CLRSCR;
WRITELN('ESTE PROGRAMA LISTA TODOS OS PRIMOS POSITIVOS MENORES QUE N.')
    WRITELN('');
    WRITELN('ESCOLHE PARA VALOR DE N UM NUMERO NATURAL MENOR QUE 32500'
    WRITELN('');
    WRITELN('FEITA A ESCOLLHA, PRESSIONA A TECLA ENTER PARA CONTINUAR')
    WRITELN('');
    WRITELN('VALOR DE N ?');
    READLN(N);
    FOR I:=0 TO N DO Y[I]:=I;
    WRITELN('ESTES SAO OS PRIMOS MENORES DO QUE N:');
    FOR I:=2 TO N DO
                  BEGIN
                      IF Y[I]<>0 THEN BEGIN
                      WRITE(Y[I]);
```

```
                        YöOÇ:=YöOÇ+1;
                        WRITE(' ');
                        FOR M:=2 TO ROUND(N/I) DO
                        YöI*MÇ:=0;
                        END;
                    END;
                WRITELN('');
                WRITELN('');
                WRITELN('EXISTEM ', YöOÇ,' PRIMOS MENORES DO QUE ',N,' ' )
END.
```

Programa decompos: Este programa faz a decomposição em fatores primos de qualquer inteiro menor do que 32500.

```
PROGRAM DECOMPOS_P;
VAR
   P,N,I,L,K:INTEGER;
   Y:ARRAYöO..32500Ç OF INTEGER;

PROCEDURE PRIMOS(N: INTEGER);
BEGIN
    FOR K:=0 TO N DO YöKÇ:=K;
    FOR K:=2 TO N DO
      IF YöKÇ<>0 THEN
          FOR I:=2 TO (N DIV K) DO YöK*IÇ:=0;
END;

BEGIN
WRITELN('O PROGRAMA DA A DECOMPOSICAO EM FATORES PRIMOS DE UM INTEITRO')
WRITELN('');
WRITELN('ESCOLHA P UM INTEIRO MENOR QUE 32500. PRESSIONE A TECLA ENTER')
WRITELN('');
WRITELN('P ?');
READLN(P);
WRITELN('');
WRITELN('ESTA E A DECOMPOSICAO EM FATORES PRIMOS DE  ',P);
PRIMOS(P);
FOR K:=2 TO  P DO
BEGIN
    IF YöKÇ<>0 THEN
    BEGIN
        YöOÇ:=0;
        IF (P MOD YöKÇ)=0 THEN
        BEGIN
        YöOÇ:=YöOÇ+1;
        L:= (P DIV YöKÇ);
        WHILE (L MOD YöKÇ) =0 DO
        BEGIN
             L:=L DIV YöKÇ;
             YöOÇ:=YöOÇ+1;
        END;
        WRITELN('          ',YöKÇ,' NA POTENCIA ',YöOÇ);
        END;
    END;
END;
END.
```

Programa Avalia: Este programa avalia um polinomio de grau N-1 nas N potencias de uma N-raiz primitiva da unidade do corpo finito Zp, com p a ser escolhido convenientemente de acordo com o

capítulo III e a raiz primitiva pode ser encontrada usando o programa seguinte.

```
PROGRAM AVALIACAO_N;
VAR
 P,N,T,I,V,W,S,Q,U,R:INTEGER;
 A,B:ARRAY[0..10000] OF INTEGER;
 C:ARRAY[0..3200] OF INTEGER;

FUNCTION POTENCIA(W,I:INTEGER): INTEGER;
BEGIN
     IF I=0 THEN Q:=1
            ELSE BEGIN
                 Q:=1;
                 S:=0;
                 REPEAT
                      Q:=Q*W;
                      S:=S+1
                  UNTIL S=I;
END;
POTENCIA:=Q;
END;

PROCEDURE EXPOENTE(N:INTEGER);
BEGIN
     A[0]:=0;
     A[1]:=(N DIV 2);
     FOR T:=1 TO (N DIV 2) DO
     BEGIN
          A[2*T]:=(A[T] DIV 2);
          A[2*T+1]:=(A[2*T] + (N DIV 2));
     END;
END;

PROCEDURE REARRANJO(V,U:INTEGER);
BEGIN
     FOR P:=0 TO V-1 DO
     BEGIN
          FOR T:=0 TO (POTENCIA(2,U)-1) DO
          BEGIN
               IF T MOD 2 =0 THEN
               C[P+(T*V)]:=B[P+(T*V)]+A[T]*B[P+(T+1)*V]
               ELSE
               C[P+(T*V)]:=B[P+(T-1)*V]+A[T]*B[P+(T*V)]
          END;
      END;
      FOR I:=0 TO 15 DO B[I]:=C[I];
 END;

BEGIN
     WRITELN('ESTE PROGRAMA AVALIA UM POLINOMIO DE GRAU N-1 EM N ');
     WRITELN(' ');
     WRITELN('PONTOS DO CORPO Zp. TAIS PONTOS SÃO OS ELEMENTOS DO');
     WRITELN(' ');
     WRITELN('CONJUNTO Xq DAS q-ESIMAS POTENCIAS DE W, UMA N-RAIZ PRI');
     WRITELN(' ');
     WRITELN('MITIVAS DA UNIDADE DE Zp. N E UMA POTENCIA DE 2 E p=N*L+1');
     WRITELN(' ');
     WRITELN('VALOR DE N ?');
     READLN(N);
     WRITELN('VALOR DE W ?');
     READLN(W);
     WRITELN('ESCREVA, UM EM CADA LINHA E EM ORDEM CRESCENTE OS COE');
     WRITELN(' ');
     WRITELN('FICIENTES DO POLINOMIO');
     FOR I:=0 TO N-1 DO READLN(B[I]);
```

```
    V:=N;
    U:=1;
    EXPOENTE(N);
    FOR I:=0 TO N-1 DO AöIÇ:=POTENCIA(W,AöIÇ);
    REPEAT
              V:=V DIV 2;
              REARRANJO(V,U);
              U:=U+1;
     UNTIL V=1;
     EXPOENTE(N);
     FOR I:=0 TO N-1 DO WRITELN('A(W NA POTENCIA ',AöIÇ:5,') = ',BöIÇ);
END.
```

Programa Primitivo: Este programa calcula o menor elemento primitivo positivo do corpo Zp, com p um primo a ser dado.

```
   PROGRAM PRIMITIVOS_P;
VAR
   P,T,N,I,L,K,M:INTEGER;
   Y:ARRAYöO..32500Ç OF INTEGER;

PROCEDURE PRIMOS(N: INTEGER);
BEGIN
    FOR K:=0 TO N DO YöKÇ:=K;
    FOR K:=2 TO N DO
      IF YöKÇ<>0 THEN
          FOR I:=2 TO (N DIV K) DO YöK*IÇ:=0;
END;

PROCEDURE POTENCIA(N,I:INTEGER);
VAR
   J,R:INTEGER;
BEGIN
J:=0;
R:=1;
REPEAT
   J:=J+1;
   R:=(R*N mod p);
   UNTIL J=I;
IF (R MOD P)=1 THEN M:=1
               ELSE M:=0;

END;

BEGIN
WRITELN('O PROGRAMA DETERMINA O MENOR  ELEMENTO PRIMITIVO DO CORPO Zp'
WRITELN('');
WRITELN('ESCOLHA p UM PRIMO MENOR QUE 32500 E PRESSIONE A TECLA ENTER
WRITELN('');
WRITELN('CERTIFIQUE-SE QUE  P ESCOLHIDO E MESMO UM PRIMO, SOB PENA DE
WRITELN('O PROGRAMA RESULTAR EM ERRO');
WRITELN('');
WRITELN('P ?');
READLN(P);
WRITELN('');
WRITELN('ESTA E A DECOMPOSICAO EM FATORES PRIMOS DE  ',P-1);
PRIMOS((P-1));
FOR K:=2 TO  (P-1) DO
BEGIN
    IF YöKÇ<>0 THEN
    BEGIN
        YöOÇ:=0;
        IF ((P-1) MOD YöKÇ)=0 THEN
                        BEGIN
                          Yö1Ç:=Yö1Ç+1;
                          YöOÇ:=YöOÇ+1;
                          L:= ((P-1) DIV YöKÇ);
                          WHILE (L MOD YöKÇ) =0 DO
                                  BEGIN
```

```
                                                L:=L DIV YöKÇ;
                                                YöOÇ:=YöOÇ+1;
                                            END;
                            WRITELN('                    ',YöKÇ,' NA POTENCIA ',YöOÇ
                            END;
   END;
END;
N:=1;
REPEAT
      L:=0;
      N:=N+1;
      FOR K:=2 TO (P-1) DO
          BEGIN
      .       IF YöKÇ<>0 THEN
                          BEGIN
                            IF ((P-1) MOD YöKÇ)=0 THEN
                                                I:=((P-1) DIV YöKÇ);
                                                POTENCIA(N,I);
                                                L:=L+M;
                          END;
          END;
UNTIL L=0 ;
WRITELN('O MENOR ELEMENTO PRIMITIVO DO CORPO DE ',P,' ELEMENTOS ');
WRITELN(' TEM VALOR  ',N);
END.
```

Programa PCR.2: Esre programa resolve problema chinês do resto de duas congruências.

```
PROGRAM PCR2;
VAR
   M,N,I,X,R,S,T,U:INTEGER;

FUNCTION INVERSO(M,N:INTEGER):INTEGER;
BEGIN
     I:=0;
     REPEAT
          I:=I+1
     UNTIL (I*M) MOD N=1
END;

PROCEDURE SOLUCAO(M,N:INTEGER);
BEGIN
     X:=INVERSO(M,N);
     T:=((S-R)*X) MOD N;
     U:=R+T*M
END;

PROCEDURE MDC(M,N:INTEGER);
BEGIN
     I:=M;
     X:=N;
     WHILE X<>0 DO
     BEGIN
          T:=X;
          X:=I-(I DIV X)*X;
          I:=T;
     END;
END;

BEGIN
     WRITELN('ESTE PROGRAMA CALCULA A SOLUCAO DO SISTEMA DE CON-',
     ');
     WRITELN('GRUENCIAS  U=R MOD M,          ONDE MDC(M,N)=1.  ',
     ');
     WRITELN('           U=S MOD N                             ',
     ');
```

```
    WRITELN('ENTRE COM O VALOR DE R');
    READLN(R);
    WRITELN('ENTRE COM O VALOR DE M');
    READLN(M);
    WRITELN('ENTRE COM O VALOR DE S');
    READLN(S);
    WRITELN('ENTRE COM O VALOR DE N');
    READLN(N);
    MDC(M,N);
    IF I<>1 THEN
    BEGIN
        WRITELN('MDC(M,N)=',I,' O PROGRAMA NAO PODE SER RODADO ');
        WRITELN('COM ESSES DADOS.');
    END
    ELSE
    BEGIN
        SOLUCAO(M,N);
        WRITELN('A MENOR SOLUCAO POSITIVA E U=',U)
   END;
END.
```

Programa FFT: Este programa computa a transformada discreta de Fourier em N pontos complexos , sendo N uma potencia de dois, via FFT.

```
PROGRAM AVALIACAO_N;
CONST
     PI=3.14159265358979;
      K=1300;
VAR
 P,N,T,I,V,S,Q,L,U,R,J:INTEGER;
 B:ARRAY[0..K,1..2] OF REAL;
 C:ARRAY[0..K,1..2] OF REAL;
 A:ARRAY[0..k,1..2] OF REAL;
 D:ARRAY[0..K,1..2] OF REAL;
 M:ARRAY[0..K] OF INTEGER;

FUNCTION POTENCIA(I,R:REAL): REAL;
BEGIN
    IF R=1 THEN
                POTENCIA:=COS(PI*I*2/N);
    IF R=2 THEN
                POTENCIA:=SIN(PI*I*2/N);

END;

PROCEDURE EXPOENTE(N:INTEGER);
BEGIN
    M[0]:=0;
    M[1]:=(N DIV 2);
    FOR T:=1 TO (N DIV 2) DO
    BEGIN
        M[2*T]:=(M[T] DIV 2);
        M[2*T+1]:=(M[2*T] + (N DIV 2));
    END;
END;

FUNCTION POT(L,I:INTEGER):INTEGER;
BEGIN
    IF I=0 THEN S:=1
    ELSE BEGIN
              Q:=0;
              S:=1;
              REPEAT
                   S:=S*L;
                   Q:=Q+1
              UNTIL Q=I;
         END;
```

```
   POT:=S;
END;


PROCEDURE REARRANJO(V,U:INTEGER);
BEGIN
     FOR P:=0 TO V-1 DO
          FOR T:=0 TO (POT(2,U)-1) DO
          BEGIN
               IF (T MOD 2)=0 THEN
               BEGIN
                    CöP+(T*V),1Ç:=BöP+(T*V),1Ç+AöT,1Ç*BöP+(T+1)*V,1Ç-
                                                        AöT,2Ç*BöP+(T+1)*V,2Ç;
                    CöP+(T*V),2Ç:=BöP+(T*V),2Ç+AöT,2Ç*BöP+(T+1)*V,1Ç+
                                                        AöT,1Ç*BöP+(T+1)*V,2Ç
               END;
               IF (T MOD 2)=1 THEN
               BEGIN
               CöP+(T*V),1Ç:=BöP+(T-1)*V,1Ç+AöT,1Ç*BöP+(T*V),1Ç-
                                                        AöT,2Ç*BöP+(T*V),2Ç;
               CöP+(T*V),2Ç:=BöP+(T-1)*V,2Ç+AöT,1Ç*BöP+(T*V),2Ç+
                                                        AöT,2Ç*BöP+(T*V),1Ç
               END;
          END;
      FOR I:=0 TO N-1 DO
      BEGIN
           BöI,1Ç:=CöI,1Ç;
           BöI,2Ç:=CöI,2Ç
      END;
 END;

BEGIN
     WRITELN('ESTE PROGRAMA CALCULA A TRNSFORMADA DISCRETA DE ',
                                                              ');
     WRITELN('FOURIER DE N PONTOS COMPLEXOS,ONDE N E POTENCIA DE',
                                                              ');
     WRITELN('DOIS. O METODO USADO NO PROGRAMA E A FFT.        ',
                                                              ');
     WRITELN('MESMO QUE O NUMERO DE AMOSTRAS NAO SEJA POTENCIA',
                                                              ');
     WRITELN('DE DOIS, ACRESCENTE ZEROS ATE COMPLETAR');
     WRITELN(' ');
     WRITELN('VALOR DE N ?');
     READLN(N);
     WRITELN('ESCREVA, UM EM CADA LINHA, A PARTE REAL DOS N PONTOS',
                                                              ');
     WRITELN('COMPLEXOS DA SEQUENCIA. ');
     FOR I:=0 TO N-1 DO READLN(BöI,1Ç);
     WRITELN('SE OS ELEMENTOS SAO REAIS,PRESSIONE O NUMERO 0',
                                                              ');
     WRITELN('SE EXISTEM ELEMENTOS COM PARTE IMAGINARIA NAO NULA',
                                                              ');
     WRITELN('PRESSIONE O NUMERO 1.');
     READLN(J);
     IF J=0 THEN FOR I:=0 TO N-1 DO BöI,2Ç:=0
     ELSE BEGIN
               WRITELN('');
               WRITELN('ESCREVA,UM EM CADA LINHA, A PARTE IMAGI-',
                                                              ');
               WRITELN('NARIA DOS ELEMENTOS DA SEQUENCIA.');
               FOR I:=0 TO N-1 DO READLN(BöI,2Ç);
          END;

     V:=N;
     U:=1;
     EXPOENTE(N);
     FOR I:=0 TO N-1 DO
      BEGIN
           DöI,1Ç:=BöI,1Ç;
           DöI,2Ç:=BöI,2Ç;
           AöI,1Ç:=POTENCIA(MöIÇ,1);
           AöI,2Ç:=POTENCIA(MöIÇ,2)
      END;
     REPEAT
               V:=V DIV 2;
```

```
            REARRANJO(V,U);
            U:=U+1;
    UNTIL V=1;
    WRITELN('                    ELEMENTO                    TRANSFOR',
                                        'MADA');
  WRITELN('POSICAO PARTE REAL IMAGINARIA        PARTE REAL  IMAGINARIA')
    FOR I:=0 TO N-1 DO WRITELN(I:4,'     ',DöI,1Ç:3:5,'     ',
                        DöI,2Ç:3:5,'     ',BöMöIÇ,1Ç:3:5,
    WRITELN('SE DESEJA IMPRIMIR, PRESSIONE O NÚMERO 1',
                                                    ');
    WRITELN('SE NAO DESEJA IMPRIMIR, PRESSIONE O NUMERO 0');
    READ(J);
    IF J=1 THEN
    BEGIN
    WRITELN(AUX,'                ELEMENTO                    TRANSFORMADA')
    WRITELN(AUX,'POSICAO PARTE REAL IMAGINARIA
    PARTE REAL  IMAGINARIA');
    FOR I:=0 TO N-1 DO WRITELN(AUX,I:4,'     ',DöI,1Ç:3:5,'     ',
                        DöI,2Ç:3:5,'     ',BöMöIÇ,1Ç:3:5,
    END;
END.
 END.
```

REFERENCIAS

AHO,A.V.,HOPCROFT,J.E. e ULLMAN,A., The Desigh and Analysis of Computer Algorihtms, Reading, Mass.,Addison-Wesley,1974.

ARSAC,J, Fourier Transforms, Englewood Cliffs, N. J., Prentice-Hall, 1966.

AUSLANDER,L e TOLMIERI,R,"Is computing with the finite Fourier transform pure or applied mathematics ?",Bull. Am. Soc., vol. 1, pp 847-897, 1979.

BERGLAND,G.D.,"A guided tour of the fast Fourier transform",IEEE Spectrum, pp 41-52, Julho 1979.

BIRKHOFF,G e MACLAINE, A Survey of Modern Algebra,ed. 4, Macmillan, 1977.

BORODIN,A,"Computational complexity-theory and practice",em A, Aho(Ed.),Currents in the Theory of Computing, Prentice-Hall, Englewood Cliffs, 1973.

BORODIN,A. e MUNRO,I.,The Computational Complexity of Algebraic and Numeric Problems, American Elsevier, New York, 1975.

COCHRAN,W.T.at ali,"Whats the fast Fourier transform", Proc. IEEE, vol.,55 pp 1664-1674, Outubro 1967.

COOLEY,J.M.,LEWIS,P.A e WELCH,P.D.,"Historical notes on the fast Fourier transform", Proc. IEEE vol. 55, pp 1675-1677,1967.

COOLEY,J.M.,LEWIS,P.A e WELCH,P.D.,"The fast Fourier transform and its applications", IEEE Trans. on Education, vol 12, pp 27-33, 1969.

COOLEY,J. M. e TUKEY,J.W.,"An algorithm for the macine calculation of complexs Fourier series", Math. Comp., vol 19, pp 297-301, 1965.

DORNHOF, L. e HOHN, F.,Applied Modern Algebra, Macmillan, New York, 1978.

ELLIOT,D. F. e RAO, K.R., Fast Transforms: Algorithms, Analysis, Applications, Academic Press, London, 1982.

ESTERMANN,E, Introduction to Modern Prime Number Theory, Cambridge University, 1952.

GENTLEMAN,W. e SANDE,G, "Fast Fourier transform for fun and profit", Proc. AFIPS, vol. 29, pp 536-568, 1966.

GILL,A., Introduction to the Theory of Finite State Machine, MacGraw-Hill, New York, 1962.

HARDY,L. e WRIGHT,T., The Theory of Numbers, ed. 4, Oxford, 1960.

HENRICI,P., Elements of Numerical Analysis, Wiley, New York, 1964.

HERSTEIN,I.N., Topics in Algebra, Blaisdell, New York, 1964.

KRONSJO, L. I., Algorithms:Their Complexity and Eficiency, Wiley, New York , 1979.

LEHMER, D. H., "The influence of computer on research in number theory", Proc. Appl. Math., vol. 20, 1973.

LIPSON, J.D.,Elements of Algebra and Algebraic Computing, Addison-Wesley, 1981.

POLLARD, J.M., "The fast Fourier transform in a finite field", Math. Comp., vol. 25, pp 365-374, 1971.