

Aula 3

Transformada rápida de Fourier

Secs 30.1 e 30.2 do CLRS e 5.6 do KT.

Produto de polinômios

Problema: Dados dois polinômios

$$a(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1} \text{ e}$$

$$b(x) = b_0 + b_1x + \cdots + b_{n-1}x^{n-1},$$

calcular o polinômio $p(x) = a(x) \cdot b(x)$.

Produto de polinômios

Problema: Dados dois polinômios

$$a(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1} \text{ e}$$

$$b(x) = b_0 + b_1x + \cdots + b_{n-1}x^{n-1},$$

calcular o polinômio $p(x) = a(x) \cdot b(x)$.

Lembre-se que $p(x) = a(x) \cdot b(x) = c_0 + c_1x + \cdots + c_{2n-2}x^{2n-2}$,
onde

$$c_k = a_0b_k + a_1b_{k-1} + \cdots + a_kb_0,$$

para $k = 0, 1, \dots, 2n - 2$.

Produto de polinômios

Problema: Dados dois polinômios

$$a(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1} \text{ e}$$

$$b(x) = b_0 + b_1x + \cdots + b_{n-1}x^{n-1},$$

calcular o polinômio $p(x) = a(x) \cdot b(x)$.

Lembre-se que $p(x) = a(x) \cdot b(x) = c_0 + c_1x + \cdots + c_{2n-2}x^{2n-2}$,
onde

$$c_k = a_0b_k + a_1b_{k-1} + \cdots + a_kb_0,$$

para $k = 0, 1, \dots, 2n - 2$.

O vetor c é a **convolução** de a e b , às vezes denotada por $a \otimes b$.

Produto de polinômios

De novo há um algoritmo $O(n^2)$ óbvio para calcular $p(x)$,
ou seja, para calcular $c_0, c_1, \dots, c_{2n-2}$.

Produto de polinômios

De novo há um algoritmo $O(n^2)$ óbvio para calcular $p(x)$,
ou seja, para calcular $c_0, c_1, \dots, c_{2n-2}$.

Queremos algo melhor!

Produto de polinômios

De novo há um algoritmo $O(n^2)$ óbvio para calcular $p(x)$,
ou seja, para calcular $c_0, c_1, \dots, c_{2n-2}$.

Queremos algo melhor! Queremos um algoritmo $O(n \lg n)$!

Produto de polinômios

De novo há um algoritmo $O(n^2)$ óbvio para calcular $p(x)$,
ou seja, para calcular $c_0, c_1, \dots, c_{2n-2}$.

Queremos algo melhor! Queremos um algoritmo $O(n \lg n)$!

Qual é a ideia do algoritmo?

Produto de polinômios

De novo há um algoritmo $O(n^2)$ óbvio para calcular $p(x)$,
ou seja, para calcular $c_0, c_1, \dots, c_{2n-2}$.

Queremos algo melhor! Queremos um algoritmo $O(n \lg n)$!

Qual é a ideia do algoritmo?

Usar uma representação alternativa para polinômios.

Produto de polinômios

De novo há um algoritmo $O(n^2)$ óbvio para calcular $p(x)$, ou seja, para calcular $c_0, c_1, \dots, c_{2n-2}$.

Queremos algo melhor! Queremos um algoritmo $O(n \lg n)$!

Qual é a ideia do algoritmo?

Usar uma representação alternativa para polinômios.

Dados n pares $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$ com $x_0, \dots, x_{n-1}$ distintos, existe um único polinômio $q(x)$ de grau menor que n tal que $q(x_i) = y_i$ para $i = 0, \dots, n-1$.

Produto de polinômios

De novo há um algoritmo $O(n^2)$ óbvio para calcular $p(x)$, ou seja, para calcular $c_0, c_1, \dots, c_{2n-2}$.

Queremos algo melhor! Queremos um algoritmo $O(n \lg n)$!

Qual é a ideia do algoritmo?

Usar uma representação alternativa para polinômios.

Dados n pares $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$ com $x_0, \dots, x_{n-1}$ distintos, existe um único polinômio $q(x)$ de grau menor que n tal que $q(x_i) = y_i$ para $i = 0, \dots, n-1$.

Por exemplo, existe uma única reta que passa por dois pontos.

Três pontos não colineares determinam uma única parábola.

Prova dessa afirmação

Dados n pares $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$ com $x_0, \dots, x_{n-1}$ distintos, existe um único polinômio $q(x)$ de grau menor que n tal que $q(x_i) = y_i$ para $i = 0, \dots, n-1$.

Prova dessa afirmação

Dados n pares $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$ com $x_0, \dots, x_{n-1}$ distintos, existe um único polinômio $q(x)$ de grau menor que n tal que $q(x_i) = y_i$ para $i = 0, \dots, n-1$.

Prova: Sejam $q_0, \dots, q_{n-1}$ os coeficientes do polinômio q .

Prova dessa afirmação

Dados n pares $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$ com $x_0, \dots, x_{n-1}$ distintos, existe um único polinômio $q(x)$ de grau menor que n tal que $q(x_i) = y_i$ para $i = 0, \dots, n-1$.

Prova: Sejam $q_0, \dots, q_{n-1}$ os coeficientes do polinômio q .

Basta notarmos que o sistema

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} q_0 \\ q_1 \\ \vdots \\ q_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

nas variáveis $q_0, q_1, \dots, q_{n-1}$ tem **solução única**.

Prova dessa afirmação

Dados n pares $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$ com $x_0, \dots, x_{n-1}$ distintos, existe um único polinômio $q(x)$ de grau menor que n tal que $q(x_i) = y_i$ para $i = 0, \dots, n-1$.

Prova: De fato, seja

$$V(x_0, x_1, \dots, x_{n-1}) = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix}.$$

Prova dessa afirmação

Dados n pares $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$ com $x_0, \dots, x_{n-1}$ distintos, existe um único polinômio $q(x)$ de grau menor que n tal que $q(x_i) = y_i$ para $i = 0, \dots, n-1$.

Prova: De fato, seja

$$V(x_0, x_1, \dots, x_{n-1}) = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix}.$$

Essa matriz é conhecida como **matriz de Vandermond**, e seu **determinante** é

$$\prod_{0 \leq j < k \leq n-1} (x_k - x_j)$$

que é **não nulo** pois os $x_0, \dots, x_{n-1}$ são distintos.

Prova dessa afirmação

Dados n pares $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$ com $x_0, \dots, x_{n-1}$ distintos, existe um único polinômio $q(x)$ de grau menor que n tal que $q(x_i) = y_i$ para $i = 0, \dots, n-1$.

Prova: De fato, seja

$$V(x_0, x_1, \dots, x_{n-1}) = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix}.$$

Essa matriz é conhecida como **matriz de Vandermond**, e seu **determinante** é

$$\prod_{0 \leq j < k \leq n-1} (x_k - x_j)$$

que é **não nulo** pois os $x_0, \dots, x_{n-1}$ são distintos.

Então $q = V(x_0, x_1, \dots, x_{n-1})^{-1}y$.



Observações que não vamos usar

Dados n pares $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$ com $x_0, \dots, x_{n-1}$ distintos, existe um único polinômio $q(x)$ de grau menor que n tal que $q(x_i) = y_i$ para $i = 0, \dots, n-1$.

Seja

$$V(x_0, x_1, \dots, x_{n-1}) = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix}$$

e $q = V(x_0, x_1, \dots, x_{n-1})^{-1}y$.

Observações que não vamos usar

Dados n pares $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$ com $x_0, \dots, x_{n-1}$ distintos, existe um único polinômio $q(x)$ de grau menor que n tal que $q(x_i) = y_i$ para $i = 0, \dots, n-1$.

Seja

$$V(x_0, x_1, \dots, x_{n-1}) = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix}$$

e $q = V(x_0, x_1, \dots, x_{n-1})^{-1}y$.

A fórmula acima dá um algoritmo $O(n^3)$ para calcular q .

Observações que não vamos usar

Dados n pares $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$ com $x_0, \dots, x_{n-1}$ distintos, existe um único polinômio $q(x)$ de grau menor que n tal que $q(x_i) = y_i$ para $i = 0, \dots, n-1$.

Seja

$$V(x_0, x_1, \dots, x_{n-1}) = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix}$$

e $q = V(x_0, x_1, \dots, x_{n-1})^{-1}y$.

A fórmula acima dá um algoritmo $O(n^3)$ para calcular q .

A **fórmula de Lagrange** dá um algoritmo $O(n^2)$ para calcular q :

$$q(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)}$$

Representação alternativa de polinômios

Dados n pares $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$ com $x_0, \dots, x_{n-1}$ distintos, existe um único polinômio $q(x)$ de grau menor que n tal que $q(x_i) = y_i$ para $i = 0, \dots, n-1$.

Representações alternativas de polinômios de grau $n-1$:

seus n coeficientes, ou

seu valor em n pontos distintos.

Representação alternativa de polinômios

Dados n pares $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$ com $x_0, \dots, x_{n-1}$ distintos, existe um único polinômio $q(x)$ de grau menor que n tal que $q(x_i) = y_i$ para $i = 0, \dots, n-1$.

Representações alternativas de polinômios de grau $n-1$:

seus n coeficientes, ou

seu valor em n pontos distintos.

O algoritmo $O(n \lg n)$ usará estas duas representações!

Representação alternativa de polinômios

Dados n pares $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$ com $x_0, \dots, x_{n-1}$ distintos, existe um único polinômio $q(x)$ de grau menor que n tal que $q(x_i) = y_i$ para $i = 0, \dots, n-1$.

Representações alternativas de polinômios de grau $n-1$:

seus n coeficientes, ou

seu valor em n pontos distintos.

O algoritmo $O(n \lg n)$ usará estas duas representações!

Lembre-se que o grau do polinômio produto é $2n-2$.

Ideia do algoritmo $O(n \lg n)$

Entrada: $a = (a_0, \dots, a_{n-1})$ e $b = (b_0, \dots, b_{n-1})$.

Obter pares

$$(x_0, y_0^a), \dots, (x_{2n-2}, y_{2n-2}^a) \text{ e } (x_0, y_0^b), \dots, (x_{2n-2}, y_{2n-2}^b)$$

onde $x_i \neq x_j$ para $i \neq j$, e

$$y_i^a = a(x_i) \text{ e } y_i^b = b(x_i) \text{ para } i = 0, \dots, 2n-2.$$

Ideia do algoritmo $O(n \lg n)$

Entrada: $a = (a_0, \dots, a_{n-1})$ e $b = (b_0, \dots, b_{n-1})$.

Obter pares

$$(x_0, y_0^a), \dots, (x_{2n-2}, y_{2n-2}^a) \text{ e } (x_0, y_0^b), \dots, (x_{2n-2}, y_{2n-2}^b)$$

onde $x_i \neq x_j$ para $i \neq j$, e

$$y_i^a = a(x_i) \text{ e } y_i^b = b(x_i) \text{ para } i = 0, \dots, 2n-2.$$

Obter pares $(x_0, q_0), \dots, (x_{2n-2}, q_{2n-2})$

onde $q_i = y_i^a \cdot y_i^b$ para $i = 0, \dots, 2n-2$.

Ideia do algoritmo $O(n \lg n)$

Entrada: $a = (a_0, \dots, a_{n-1})$ e $b = (b_0, \dots, b_{n-1})$.

Obter pares

$$(x_0, y_0^a), \dots, (x_{2n-2}, y_{2n-2}^a) \text{ e } (x_0, y_0^b), \dots, (x_{2n-2}, y_{2n-2}^b)$$

onde $x_i \neq x_j$ para $i \neq j$, e

$$y_i^a = a(x_i) \text{ e } y_i^b = b(x_i) \text{ para } i = 0, \dots, 2n-2.$$

Obter pares $(x_0, q_0), \dots, (x_{2n-2}, q_{2n-2})$

onde $q_i = y_i^a \cdot y_i^b$ para $i = 0, \dots, 2n-2$.

Determinar $q(x)$ tal que $q(x_i) = q_i$ para $i = 0, \dots, 2n-2$.

Ideia do algoritmo $O(n \lg n)$

Entrada: $a = (a_0, \dots, a_{n-1})$ e $b = (b_0, \dots, b_{n-1})$.

Obter pares

$$(x_0, y_0^a), \dots, (x_{2n-2}, y_{2n-2}^a) \text{ e } (x_0, y_0^b), \dots, (x_{2n-2}, y_{2n-2}^b)$$

onde $x_i \neq x_j$ para $i \neq j$, e

$$y_i^a = a(x_i) \text{ e } y_i^b = b(x_i) \text{ para } i = 0, \dots, 2n-2.$$

Obter pares $(x_0, q_0), \dots, (x_{2n-2}, q_{2n-2})$

onde $q_i = y_i^a \cdot y_i^b$ para $i = 0, \dots, 2n-2$.

Determinar $q(x)$ tal que $q(x_i) = q_i$ para $i = 0, \dots, 2n-2$.

O passo do meio consome tempo $O(n)$ trivialmente.

Ideia do algoritmo $O(n \lg n)$

Dado $a = (a_0, \dots, a_{n-1})$,
como calcular o valor de a em $2n - 1$ pontos em $O(n \lg n)$?

Ideia do algoritmo $O(n \lg n)$

Dado $a = (a_0, \dots, a_{n-1})$,
como calcular o valor de a em $2n - 1$ pontos em $O(n \lg n)$?

Dado x , quanto tempo levamos para calcular $a(x)$?

Ideia do algoritmo $O(n \lg n)$

Dado $a = (a_0, \dots, a_{n-1})$,
como calcular o valor de a em $2n - 1$ pontos em $O(n \lg n)$?

Dado x , quanto tempo levamos para calcular $a(x)$? $\Theta(n)$.

Ideia do algoritmo $O(n \lg n)$

Dado $a = (a_0, \dots, a_{n-1})$,
como calcular o valor de a em $2n - 1$ pontos em $O(n \lg n)$?

Dado x , quanto tempo levamos para calcular $a(x)$? $\Theta(n)$.

Então se escolhermos $x_0, \dots, x_{2n-2}$ arbitrariamente,
levaremos tempo $\Theta(n^2)$ nesta etapa.

Ideia do algoritmo $O(n \lg n)$

Dado $a = (a_0, \dots, a_{n-1})$,
como calcular o valor de a em $2n - 1$ pontos em $O(n \lg n)$?

Dado x , quanto tempo levamos para calcular $a(x)$? $\Theta(n)$.

Então se escolhermos $x_0, \dots, x_{2n-2}$ arbitrariamente,
levaremos tempo $\Theta(n^2)$ nesta etapa.

Que pontos escolher então?

Ideia do algoritmo $O(n \lg n)$

Dado $a = (a_0, \dots, a_{n-1})$,

como calcular o valor de a em $2n - 1$ pontos em $O(n \lg n)$?

Dado x , quanto tempo levamos para calcular $a(x)$? $\Theta(n)$.

Então se escolhermos $x_0, \dots, x_{2n-2}$ arbitrariamente, levaremos tempo $\Theta(n^2)$ nesta etapa.

Que pontos escolher então?

Pontos muito especiais... as raízes da unidade!

Ideia do algoritmo $O(n \lg n)$

Dado $a = (a_0, \dots, a_{n-1})$,
como calcular o valor de a em $2n - 1$ pontos em $O(n \lg n)$?

Dado x , quanto tempo levamos para calcular $a(x)$? $\Theta(n)$.

Então se escolhermos $x_0, \dots, x_{2n-2}$ arbitrariamente,
levaremos tempo $\Theta(n^2)$ nesta etapa.

Que pontos escolher então?

Pontos muito especiais... as raízes da unidade!

Quem são estas??

Raízes da unidade

São definidas para cada n .

Raízes da unidade

São definidas para cada n .

As raízes n -ésimas da unidade são as raízes da equação

$$x^n = 1.$$

(São números complexos! Lembra deles?)

Raízes da unidade

São definidas para cada n .

As **raízes n -ésimas da unidade** são as raízes da equação

$$x^n = 1.$$

(São números complexos! Lembra deles?)

Existem exatamente n raízes da unidade.

Raízes da unidade

São definidas para cada n .

As raízes n -ésimas da unidade são as raízes da equação

$$x^n = 1.$$

(São números complexos! Lembra deles?)

Existem exatamente n raízes da unidade.

Seja $\omega_n = e^{2\pi i/n} = \cos(2\pi/n) + i \sin(2\pi/n)$.

Raízes n -ésimas da unidade: para $k = 0, 1, \dots, n-1$,

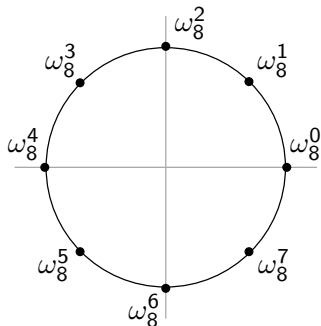
$$\omega_n^k = e^{2\pi k i/n}.$$

Raízes da unidade

Raízes n -ésimas da unidade: para $k = 0, 1, \dots, n - 1$,

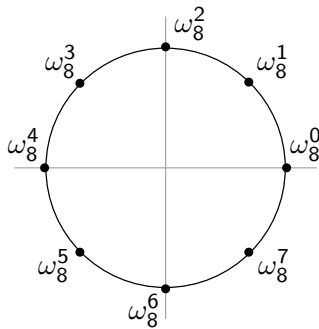
$$\omega_n^k = e^{2\pi ki/n}.$$

Para $n = 8$ temos



Raízes da unidade

Para $n = 8$ temos



$$\begin{aligned}\omega_8^0 &= e^0 = 1 \\ \omega_8^1 &= e^{\pi i/4} = \cos(\pi/4) + i \sin(\pi/4) \\ \omega_8^2 &= e^{\pi i/2} = \cos(\pi/2) + i \sin(\pi/2) = i \\ \omega_8^3 &= e^{3\pi i/4} = \cos(3\pi/4) + i \sin(3\pi/4) \\ \omega_8^4 &= e^{\pi i} = \cos(\pi) + i \sin(\pi) = -1 \\ \omega_8^5 &= e^{5\pi i/4} = \cos(5\pi/4) + i \sin(5\pi/4) \\ \omega_8^6 &= e^{3\pi i/2} = \cos(3\pi/2) + i \sin(3\pi/2) = -i \\ \omega_8^7 &= e^{7\pi i/4} = \cos(7\pi/4) + i \sin(7\pi/4)\end{aligned}$$

Transformada discreta de Fourier

Seja $\omega_n = e^{2\pi i/n}$.

Raízes n -ésimas da unidade: para $k = 0, 1, \dots, n-1$,

$$\omega_n^k = e^{2\pi ki/n}.$$

Dado um vetor $a = (a_0, a_1, \dots, a_{n-1})$, representando os coeficientes de um polinômio que denotamos por $a(x)$, a **transformada discreta de Fourier** (DFT) de ordem n de a é o vetor $y = (y_0, y_1, \dots, y_{n-1})$ onde $y_k = a(\omega_n^k)$ para $k = 0, 1, \dots, n-1$.

Objetivo: algoritmo que, dado um vetor $a = (a_0, \dots, a_{n-1})$, determina a sua DFT de ordem n em tempo $\Theta(n \lg n)$.

Voltando ao produto de polinômios...

Lembre-se... queremos...

Obter pares

$(x_0, y_0^a), \dots, (x_{2n-2}, y_{2n-2}^a)$ e $(x_0, y_0^b), \dots, (x_{2n-2}, y_{2n-2}^b)$
onde $x_i \neq x_j$ para $i \neq j$, e

$y_i^a = a(x_i)$ e $y_i^b = b(x_i)$ para $i = 0, \dots, 2n-2$.

Obter pares $(x_0, q_0), \dots, (x_{2n-2}, q_{2n-2})$

onde $q_i = y_i^a \cdot y_i^b$ para $i = 0, \dots, 2n-2$.

Determinar $q(x)$ tal que $q(x_i) = q_i$ para $i = 0, \dots, 2n-2$.

Voltando ao produto de polinômios...

Lembre-se... queremos...

Obter pares

$(x_0, y_0^a), \dots, (x_{2n-2}, y_{2n-2}^a)$ e $(x_0, y_0^b), \dots, (x_{2n-2}, y_{2n-2}^b)$
onde $x_i \neq x_j$ para $i \neq j$, e

$y_i^a = a(x_i)$ e $y_i^b = b(x_i)$ para $i = 0, \dots, 2n-2$.

Obter pares $(x_0, q_0), \dots, (x_{2n-2}, q_{2n-2})$

onde $q_i = y_i^a \cdot y_i^b$ para $i = 0, \dots, 2n-2$.

Determinar $q(x)$ tal que $q(x_i) = q_i$ para $i = 0, \dots, 2n-2$.

Primeira etapa:

dados vetores $a = (a_0, \dots, a_{n-1})$ e $b = (b_0, \dots, b_{n-1})$,
estendemos tais vetores adicionando n zeros em cada um,
obtendo $a = (a_0, \dots, a_{2n-1})$ e $b = (b_0, \dots, b_{2n-1})$,
e calculamos $\text{DFT}_{2n}(a)$ e $\text{DFT}_{2n}(b)$.

Transformada rápida de Fourier

Como calcular a $\text{DFT}_{2n}(a)$ em tempo $O(n \lg n)$?

Transformada rápida de Fourier

Como calcular a $\text{DFT}_{2n}(a)$ em tempo $O(n \lg n)$?

Por divisão e conquista!

Transformada rápida de Fourier

Como calcular a $\text{DFT}_{2n}(a)$ em tempo $O(n \lg n)$?

Por divisão e conquista!

Considere $a^0(x) = a_0 + a_2x + a_4x^2 + \cdots + a_{2n-2}x^{n-1}$ e
 $a^1(x) = a_1 + a_3x + a_5x^2 + \cdots + a_{2n-1}x^{n-1}$.

Transformada rápida de Fourier

Como calcular a $\text{DFT}_{2n}(a)$ em tempo $O(n \lg n)$?

Por divisão e conquista!

Considere $a^0(x) = a_0 + a_2x + a_4x^2 + \cdots + a_{2n-2}x^{n-1}$ e
 $a^1(x) = a_1 + a_3x + a_5x^2 + \cdots + a_{2n-1}x^{n-1}$.

Observe que $a(x) = a^0(x^2) + xa^1(x^2)$.

Transformada rápida de Fourier

Como calcular a $\text{DFT}_{2n}(a)$ em tempo $O(n \lg n)$?

Por divisão e conquista!

Considere $a^0(x) = a_0 + a_2x + a_4x^2 + \cdots + a_{2n-2}x^{n-1}$ e
 $a^1(x) = a_1 + a_3x + a_5x^2 + \cdots + a_{2n-1}x^{n-1}$.

Observe que $a(x) = a^0(x^2) + xa^1(x^2)$.

Com isso, calcular $a(\omega_{2n}^k)$ para $k = 0, \dots, 2n - 1$
reduz-se a calcular a^0 e a^1 em $(\omega_{2n}^k)^2 = \omega_{2n}^{2k}$ para $k = 0, \dots, 2n - 1$.

Transformada rápida de Fourier

Como calcular a $\text{DFT}_{2n}(a)$ em tempo $O(n \lg n)$?

Por divisão e conquista!

Considere $a^0(x) = a_0 + a_2x + a_4x^2 + \cdots + a_{2n-2}x^{n-1}$ e
 $a^1(x) = a_1 + a_3x + a_5x^2 + \cdots + a_{2n-1}x^{n-1}$.

Observe que $a(x) = a^0(x^2) + x a^1(x^2)$.

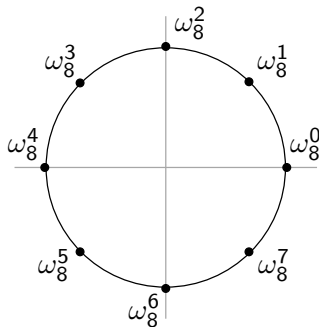
Com isso, calcular $a(\omega_{2n}^k)$ para $k = 0, \dots, 2n - 1$
reduz-se a calcular a^0 e a^1 em $(\omega_{2n}^k)^2 = \omega_{2n}^{2k}$ para $k = 0, \dots, 2n - 1$.

Beleza das raízes da unidade: os quadrados das raízes de ordem $2n$
são exatamente as raízes de ordem n !

Cada raiz de ordem n aparece duas vezes.

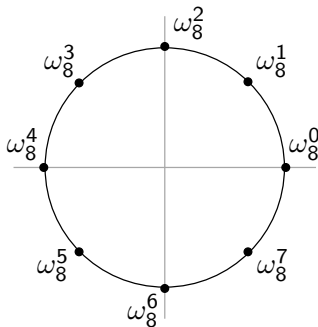
Exemplo

Quem são os quadrados das raízes de ordem $n = 8$?



Exemplo

Quem são os quadrados das raízes de ordem $n = 8$?

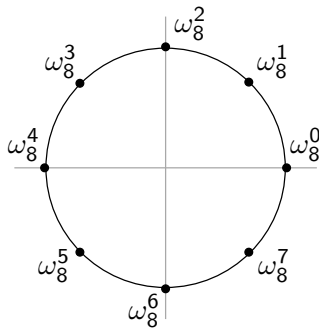


Quadrado das raízes $2n$ -ésimas da unidade: para $k = 0, 1, \dots, 2n - 1$,

$$\omega_{2n}^{2k} = e^{2\pi 2ki/2n} = e^{2\pi ki/n} = \omega_n^k.$$

Raízes da unidade

Para $n = 8$ temos



$$\begin{aligned}\omega_8^0 &= e^0 = 1 \\ \omega_8^1 &= e^{\pi i/4} = \cos(\pi/4) + i \sin(\pi/4) \\ \omega_8^2 &= e^{\pi i/2} = \cos(\pi/2) + i \sin(\pi/2) = i \\ \omega_8^3 &= e^{3\pi i/4} = \cos(3\pi/4) + i \sin(3\pi/4) \\ \omega_8^4 &= e^{\pi i} = \cos(\pi) + i \sin(\pi) = -1 \\ \omega_8^5 &= e^{5\pi i/4} = \cos(5\pi/4) + i \sin(5\pi/4) \\ \omega_8^6 &= e^{3\pi i/2} = \cos(3\pi/2) + i \sin(3\pi/2) = -i \\ \omega_8^7 &= e^{7\pi i/4} = \cos(7\pi/4) + i \sin(7\pi/4)\end{aligned}$$

Raízes da unidade

Propriedades:

$$\omega_{2n}^{2k} = e^{2\pi i(2k)/(2n)} = e^{2\pi i k/n} = \omega_n^k$$

Raízes da unidade

Propriedades:

$$\omega_{2n}^{2k} = e^{2\pi i(2k)/(2n)} = e^{2\pi i k/n} = \omega_n^k$$

$$(\omega_{2n}^{k+n})^2 = \omega_{2n}^{2k} \cdot \omega_{2n}^{2n} = \omega_{2n}^{2k} = \omega_n^k$$

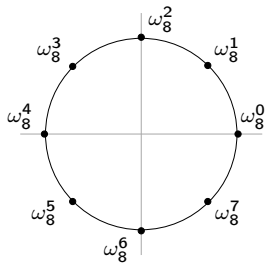
Raízes da unidade

Propriedades:

$$\omega_{2n}^{2k} = e^{2\pi i(2k)/(2n)} = e^{2\pi i k/n} = \omega_n^k$$

$$(\omega_{2n}^{k+n})^2 = \omega_{2n}^{2k} \cdot \omega_{2n}^{2n} = \omega_{2n}^{2k} = \omega_n^k$$

$$\omega_{2n}^{k+n} = -\omega_{2n}^k$$



Raízes da unidade

Propriedades:

$$\omega_{2n}^{2k} = e^{2\pi i(2k)/(2n)} = e^{2\pi i k/n} = \omega_n^k$$

$$(\omega_{2n}^{k+n})^2 = \omega_{2n}^{2k} \cdot \omega_{2n}^{2n} = \omega_{2n}^{2k} = \omega_n^k$$

$$\omega_{2n}^{k+n} = -\omega_{2n}^k$$

Algoritmo de divisão e conquista para calcular $\text{DFT}_{2n}(a)$:

Divisão: Dado $a = (a_0, \dots, a_{2n-1})$, montar a^0 e a^1 .

Lembre-se que $a(x) = a_0(x^2) + xa_1(x^2)$.

Raízes da unidade

Propriedades:

$$\omega_{2n}^{2k} = e^{2\pi i(2k)/(2n)} = e^{2\pi i k/n} = \omega_n^k$$

$$(\omega_{2n}^{k+n})^2 = \omega_{2n}^{2k} \cdot \omega_{2n}^{2n} = \omega_{2n}^{2k} = \omega_n^k$$

$$\omega_{2n}^{k+n} = -\omega_{2n}^k$$

Algoritmo de divisão e conquista para calcular $\text{DFT}_{2n}(a)$:

Divisão: Dado $a = (a_0, \dots, a_{2n-1})$, montar a^0 e a^1 .

Lembre-se que $a(x) = a_0(x^2) + xa_1(x^2)$.

Conquista: Recursivamente calcular $y^0 = \text{DFT}_n(a^0)$ e $y^1 = \text{DFT}_n(a^1)$.

Raízes da unidade

Propriedades:

$$\omega_{2n}^{2k} = e^{2\pi i(2k)/(2n)} = e^{2\pi i k/n} = \omega_n^k$$

$$(\omega_{2n}^{k+n})^2 = \omega_{2n}^{2k} \cdot \omega_{2n}^{2n} = \omega_{2n}^{2k} = \omega_n^k$$

$$\omega_{2n}^{k+n} = -\omega_{2n}^k$$

Algoritmo de divisão e conquista para calcular $\text{DFT}_{2n}(a)$:

Divisão: Dado $a = (a_0, \dots, a_{2n-1})$, montar a^0 e a^1 .

Lembre-se que $a(x) = a_0(x^2) + x a_1(x^2)$.

Conquista: Recursivamente calcular $y^0 = \text{DFT}_n(a^0)$ e $y^1 = \text{DFT}_n(a^1)$.

Combinação: Para $k = 0, \dots, n-1$,
calcular $y_k = y_k^0 + \omega_{2n}^k y_k^1$ e $y_{n+k} = y_k^0 - \omega_{2n}^k y_k^1$.

$$y_k = a(\omega_{2n}^k) = a_0(\omega_{2n}^{2k}) + \omega_{2n}^k a_1(\omega_{2n}^{2k}) = a_0(\omega_n^k) + \omega_{2n}^k a_1(\omega_n^k) = y_k^0 + \omega_{2n}^k y_k^1.$$

Transformada rápida de Fourier

DFT (a, n)

▷ n é uma potência de 2

```
1  se  $n = 1$ 
2      então devolva  $a$ 
3   $a^0 \leftarrow (a_0, a_2, \dots, a_{n-2})$ 
4   $a^1 \leftarrow (a_1, a_3, \dots, a_{n-1})$ 
5   $y^0 \leftarrow \text{DFT}(a^0, n/2)$ 
6   $y^1 \leftarrow \text{DFT}(a^1, n/2)$ 
7   $\omega_n \leftarrow e^{2\pi i/n}$ 
8   $\omega \leftarrow 1$ 
9  para  $k \leftarrow 0$  até  $n/2 - 1$  faça
10       $y[k] \leftarrow y^0[k] + \omega y^1[k]$ 
11       $y[k + n/2] \leftarrow y^0[k] - \omega y^1[k]$ 
12       $\omega \leftarrow \omega \omega_n$ 
13  devolva  $y$ 
```

Consumo de tempo

O consumo de tempo do algoritmo é dado pela recorrência

$$T(n) = 2 T(n/2) + O(n).$$

Portanto é $O(n \lg n)$.

Voltando ao produto de polinômios...

Queremos...

Para $x_0, \dots, x_{2n-2}$ distintos, calcular

$y_i^a = a(x_i)$ e $y_i^b = b(x_i)$ para $i = 0, \dots, 2n-2$.

Calcular $q_i = y_i^a \cdot y_i^b$ para $i = 0, \dots, 2n-2$.

Determinar $q(x)$ tal que $q(x_i) = q_i$ para $i = 0, \dots, 2n-2$.

Voltando ao produto de polinômios...

Queremos...

Para $x_0, \dots, x_{2n-2}$ distintos, calcular

$y_i^a = a(x_i)$ e $y_i^b = b(x_i)$ para $i = 0, \dots, 2n-2$.

Calcular $q_i = y_i^a \cdot y_i^b$ para $i = 0, \dots, 2n-2$.

Determinar $q(x)$ tal que $q(x_i) = q_i$ para $i = 0, \dots, 2n-2$.

Primeira etapa:

dados vetores $a = (a_0, \dots, a_{n-1})$ e $b = (b_0, \dots, b_{n-1})$,
estendemos tais vetores adicionando n zeros em cada um,
obtendo $\tilde{a} = (a_0, \dots, a_{2n-1})$ e $\tilde{b} = (b_0, \dots, b_{2n-1})$,
e calculamos $y^a = \text{FFT}_{2n}(\tilde{a})$ e $y^b = \text{FFT}_{2n}(\tilde{b})$.

Voltando ao produto de polinômios...

Queremos...

Para $x_0, \dots, x_{2n-2}$ distintos, calcular

$y_i^a = a(x_i)$ e $y_i^b = b(x_i)$ para $i = 0, \dots, 2n-2$.

Calcular $q_i = y_i^a \cdot y_i^b$ para $i = 0, \dots, 2n-2$.

Determinar $q(x)$ tal que $q(x_i) = q_i$ para $i = 0, \dots, 2n-2$.

Primeira etapa:

dados vetores $a = (a_0, \dots, a_{n-1})$ e $b = (b_0, \dots, b_{n-1})$,
estendemos tais vetores adicionando n zeros em cada um,
obtendo $\tilde{a} = (a_0, \dots, a_{2n-1})$ e $\tilde{b} = (b_0, \dots, b_{2n-1})$,
e calculamos $y^a = \text{FFT}_{2n}(\tilde{a})$ e $y^b = \text{FFT}_{2n}(\tilde{b})$.

Segunda etapa: óbvia...

Voltando ao produto de polinômios...

Queremos...

Para $x_0, \dots, x_{2n-2}$ distintos, calcular

$y_i^a = a(x_i)$ e $y_i^b = b(x_i)$ para $i = 0, \dots, 2n-2$.

Calcular $q_i = y_i^a \cdot y_i^b$ para $i = 0, \dots, 2n-2$.

Determinar $q(x)$ tal que $q(x_i) = q_i$ para $i = 0, \dots, 2n-2$.

Primeira etapa:

dados vetores $a = (a_0, \dots, a_{n-1})$ e $b = (b_0, \dots, b_{n-1})$,
estendemos tais vetores adicionando n zeros em cada um,
obtendo $\tilde{a} = (a_0, \dots, a_{2n-1})$ e $\tilde{b} = (b_0, \dots, b_{2n-1})$,
e calculamos $y^a = \text{FFT}_{2n}(\tilde{a})$ e $y^b = \text{FFT}_{2n}(\tilde{b})$.

Segunda etapa: óbvia...

Terceira etapa: interpolação.

Interpolação

Primeira etapa: dado um vetor $a = (a_0, \dots, a_{n-1})$, calcular $y = (y_0, \dots, y_{n-1})$ tal que $y = V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1}) \cdot a$.

Interpolação

Primeira etapa: dado um vetor $\mathbf{a} = (a_0, \dots, a_{n-1})$, calcular $\mathbf{y} = (y_0, \dots, y_{n-1})$ tal que $\mathbf{y} = V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1}) \cdot \mathbf{a}$.

A matriz $V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1})$ é a matriz de Vandermonde para $\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1}$.

Interpolação

Primeira etapa: dado um vetor $\mathbf{a} = (a_0, \dots, a_{n-1})$, calcular $\mathbf{y} = (y_0, \dots, y_{n-1})$ tal que $\mathbf{y} = V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1}) \cdot \mathbf{a}$.

A matriz $V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1})$ é a matriz de Vandermonde para $\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1}$.

Terceira etapa: dado um vetor $\mathbf{y} = (y_0, \dots, y_{n-1})$, calcular $\mathbf{q} = (q_0, \dots, q_{n-1})$ tal que $\mathbf{q} = V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1})^{-1} \cdot \mathbf{y}$.

Interpolação

Primeira etapa: dado um vetor $\mathbf{a} = (a_0, \dots, a_{n-1})$, calcular $\mathbf{y} = (y_0, \dots, y_{n-1})$ tal que $\mathbf{y} = V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1}) \cdot \mathbf{a}$.

A matriz $V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1})$ é a matriz de Vandermonde para $\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1}$.

Terceira etapa: dado um vetor $\mathbf{y} = (y_0, \dots, y_{n-1})$, calcular $\mathbf{q} = (q_0, \dots, q_{n-1})$ tal que $\mathbf{q} = V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1})^{-1} \cdot \mathbf{y}$.

Como calcular $\mathbf{q}$ em tempo $O(n \lg n)$?

Interpolação

Primeira etapa: dado um vetor $\mathbf{a} = (a_0, \dots, a_{n-1})$, calcular $\mathbf{y} = (y_0, \dots, y_{n-1})$ tal que $\mathbf{y} = V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1}) \cdot \mathbf{a}$.

A matriz $V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1})$ é a matriz de Vandermonde para $\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1}$.

Terceira etapa: dado um vetor $\mathbf{y} = (y_0, \dots, y_{n-1})$, calcular $\mathbf{q} = (q_0, \dots, q_{n-1})$ tal que $\mathbf{q} = V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1})^{-1} \cdot \mathbf{y}$.

Como calcular $\mathbf{q}$ em tempo $O(n \lg n)$?

Teorema: A inversa da matriz de Vandermonde $V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1})$ tem na posição (j, k) o valor ω_n^{-jk} / n para $j, k = 0, 1, \dots, n-1$

Interpolação

Teorema: A inversa da matriz de Vandermonde $V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1})$ tem na posição (j, k) o valor ω_n^{-jk}/n para $j, k = 0, 1, \dots, n-1$

Ou seja,

$$V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1})^{-1} = \frac{1}{n} V(\omega_n^0, \omega_n^{-1}, \dots, \omega_n^{-(n-1)}).$$

Interpolação

Teorema: A inversa da matriz de Vandermonde $V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1})$ tem na posição (j, k) o valor ω_n^{-jk}/n para $j, k = 0, 1, \dots, n-1$

Ou seja,

$$V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1})^{-1} = \frac{1}{n} V(\omega_n^0, \omega_n^{-1}, \dots, \omega_n^{-(n-1)}).$$

Objetivo:

Modificar o algoritmo FFT para fazer a interpolação.

Inversa da DFT

Note que

$$V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1}) = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \dots & \omega_n^{2(n-1)} \\ \vdots & & & & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)^2} \end{pmatrix}$$

é uma matriz simétrica e,
como o conjugado complexo de $e^{\theta i}$ é exatamente $e^{-\theta i}$,
temos que

$$V(\omega_n^0, \omega_n^{-1}, \dots, \omega_n^{-(n-1)}) = V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1})^*$$

Inversa da DFT

Como $V(\omega_n^0, \omega_n^{-1}, \dots, \omega_n^{-(n-1)}) = V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1})^*$,
então podemos calcular

$$q = \frac{1}{n} V(\omega_n^0, \omega_n^{-1}, \dots, \omega_n^{-(n-1)}) y$$

calculando

$$q' = V(\omega_n^0, \omega_n^{-1}, \dots, \omega_n^{-(n-1)}) y$$

pelo mesmo método da primeira etapa e então $q = q'/n$.

Ou seja, podemos usar a própria função FFT
para calcular q a partir de y .

Inversa da DFT

Como $V(\omega_n^0, \omega_n^{-1}, \dots, \omega_n^{-(n-1)}) = V(\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1})^*$,
então podemos calcular

$$q = \frac{1}{n} V(\omega_n^0, \omega_n^{-1}, \dots, \omega_n^{-(n-1)}) y$$

calculando

$$q' = V(\omega_n^0, \omega_n^{-1}, \dots, \omega_n^{-(n-1)}) y$$

pelo mesmo método da primeira etapa e então $q = q'/n$.

Ou seja, podemos usar a própria função FFT
para calcular q a partir de y .

Exercício:

Modifique o algoritmo FFT para fazer a interpolação.

Aplicações de FFT

Processamento de imagens

suavização da imagem

remoção de ruído

destaque de contornos

Aplicações de FFT

Processamento de imagens

- suavização da imagem
- remoção de ruído
- destaque de contornos

Música

- equalizadores
- reconhecimento de notas

Aplicações de FFT

Processamento de imagens

- suavização da imagem
- remoção de ruído
- destaque de contornos

Música

- equalizadores
- reconhecimento de notas

Combinação de histogramas