2/27

Is there a CS professor who won an Oscar

Prof. Steve Marschner

(a) Yes (b) No

Fast Fourier Transform (FFT) and applications.

Discrete Fourier Transform. (DFT)

Defn Let $a_0, a_1, \ldots, a_{n-1}$ be a sequence of complex numbers.

$b_0, b_1, \ldots, b_{n-1}$ is called the DFT of $a_0, a_1, \ldots, a_{n-1}$, where

$$b_i = \sum_{j=0}^{n-1} a_j \omega^{i \cdot j},$$

where:

ω is a 'primitive' n^{th} root of unity.

Complex numbers and roots of unity.

$z = a + ib$, a, b are reals; $i = \sqrt{-1}$.

$$(z^{c_1})^{c_2} = z^{c_1 c_2};$$

$$z^{c_1} \cdot z^{c_2} = z^{c_1 + c_2}$$

eq^n: $x^n = 1$ over complex numbers.

$\{\omega^0, \omega, \omega^2, \omega^3, \ldots, \omega^{n-1}\}$ are the n distinct solutions to $x^n - 1 = 0$.

Intuition. ✓ signal ✓ time domain.

$$f: \{0, 1, \ldots, n-1\} \to \mathbb{C}$$

$a_0 = f(0)$, $a_1 = f(1)$, $\ldots$, $a_{n-1} = f(n-1)$

DFT of f is a function $g: \{0, 1, \ldots, n-1\} \to \mathbb{C}$

$$g(i) = \sum_{j=0}^{n-1} f(j)\, \omega^{i \cdot j}$$

Omega.

$b_0, b_1, \ldots, b_{n-1}$ is called the DFT of $a_0, a_1, \ldots, a_{n-1}$, where

$$b_i = \sum_{j=0}^{n-1} a_j\, \omega^{i \cdot j},$$

$\forall i \in \{0, \ldots, n-1\}$

$b_i = g(i)$.

Inverse DFT

Let $(b_0, \ldots, b_{n-1})$ be the DFT of $(a_0, \ldots, a_{n-1})$.

Can we recover $(a_0, \ldots, a_{n-1})$ given $(b_0, \ldots, b_{n-1})$?

Thm: Let $b_0, \ldots, b_{n-1}$ be the DFT of $a_0, \ldots, a_{n-1}$. Then,

$$a_j = \frac{1}{n} \sum_{i=0}^{n-1} b_i \, \omega^{-i \cdot j}$$

Claim: For all integers $j, k \in \{0, 1, \ldots, n-1\}$,

$$\sum_{i=0}^{n-1} \omega^{i \cdot j} \, \omega^{-i \cdot k} = n \cdot \delta_{j,k},$$

where $\delta_{j,k} = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{o/w.} \end{cases}$

Pf of claim: If $j = k$, obvious ✓.

Let $j \neq k$.

$$\sum_{i=0}^{n-1} \left(\omega^{(j-k)}\right)^i = \frac{\left(\omega^{(j-k)}\right)^n - 1}{\omega^{j-k} - 1}$$

(Geometric Progression)

$$= \frac{1-1}{\omega^{j-k}-1} = 0 .$$

$$\left(\omega^{(j-k)}\right)^n = \omega^{n\cdot(j-k)} = \left(\omega^n\right)^{j-k}$$

$$= 1^{j-k} = 1 .$$

□

Proof of IDFT:

$$\frac{1}{n}\sum_{i=0}^{n-1} b_i\, \omega^{-ij} = \frac{1}{n}\sum_{i=0}^{n-1}\left(\sum_{l=0}^{n-1} a_l\, \omega^{il}\right)\cdot \omega^{-ij}$$

$$= \frac{1}{n}\sum_{l=0}^{n-1} a_l \left(\sum_{i=0}^{n-1} \omega^{il}\cdot\omega^{-ij}\right)$$

$$= \frac{1}{n}\sum_{l=0}^{n-1} a_l \cdot n\cdot \delta_{l,j}$$

$$= a_j .$$

□

Computing the DFT

Input: $a_0, a_1, \cdots, a_{n-1}$

Compute DFT: $b_0, \cdots, b_{n-1}$.

Recall, $$b_i = \sum_{j=0}^{n-1} a_j \omega^{i \cdot j}, \quad i = 0, 1, \cdots, n-1.$$

Obvious strategy: $O(n^2)$ time.

Define: $$p(x) = \sum_{j=0}^{n-1} a_j x^j.$$

Notice that $b_i = p(\omega^i)$, $i = 0, \cdots, n-1$.

Assume: n is a power of 2.

$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$.

$$p(x) = \underbrace{a_0 + a_2 x^2 + a_4 x^4 + \cdots + a_{n-2} x^{n-2}}_{P_{even}(x^2)} + x\underbrace{\left(a_1 + a_3 x^2 + a_5 x^4 + \cdots + a_{n-1} x^{n-2} \right)}_{P_{odd}(x^2)}$$

Define: $P_{even}(y) = a_0 + a_2 y + a_4 y^2 + \cdots + a_{n-2} y^{\frac{n-2}{2}}$

$P_{odd}(y) = a_1 + a_3 y + a_5 y^2 + \cdots + a_{n-1} y^{\frac{n-2}{2}}$.

* $p(x) = P_{even}(x^2) + x \cdot P_{odd}(x^2)$.

Obs. $\deg(P_{even}), \deg(P_{odd}) \leq n/2$.

For any $j \in \{0, \ldots, n-1\}$,

$$p(\omega^j) = p_{even}(\omega^{2j}) + \omega^j p_{odd}(\omega^{2j})$$

Thus, a divide-and-conquer strategy would be to evaluate p_{even}, p_{odd} at all $\left(\frac{n}{2}\right)^{th}$ roots of unity.

Given this, it takes $O(n)$ time to compute $p(\omega^j)$, $j = 0, 1, \ldots, n-1$.

Thus, we have

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

$$T(n) = O(n \log n).$$

→ We have assumed that we can do arbitrary precision floating point arithmetic in $O(1)$ time. In practice, it turns out that FFT is extremely well-behaved with respect to rounding-off errors.

3/1

Recall: given $a_0, a_1, \ldots, a_{n-1}$

we can compute DFT of $(a_0, \ldots, a_{n-1})$, given by $(b_0, \ldots, b_{n-1})$ in $O(n \log n)$ time, where $\forall i = 0, \ldots, n-1$,

$$b_i = \sum_{j=0}^{n-1} a_j \omega^{ij},$$

ω is a primitive n^{th} root of unity.

$$a_j = \frac{1}{n} \sum_{i=0}^{n-1} b_i \omega^{-ij}.$$

Computing Convolutions.

Def^n: Let $(a_0, \ldots, a_{n-1})$ and $(b_0, \ldots, b_{n-1})$ be two sequences of complex numbers.

The convolution of the sequences is given by $(e_0, e_1, \ldots, e_{2n-2})$,

where

$$e_l = \sum_{(i,j):\, i+j=l} a_i b_j$$

Algorithm to compute convolutions.

$l = 2n - 1$.

$(a_0, \ldots, a_{n-1}) \xrightarrow{\text{Padded with 0's}} (a_0, \ldots, a_{n-1}, \underbrace{0, 0, \ldots}_{l})$ (a_m marks the first padded 0)

$(b_0, \ldots, b_{n-1}) \xrightarrow{\text{"}} (b_0, \ldots, b_{n-1}, 0, 0, \ldots)$

Notation: $a_i = 0$, $i > n-1$.

$b_i = 0$, $i > n-1$.

$(a_0, \ldots, a_{2n-2}) \xrightarrow{DFT} (c_0, \ldots, c_{2n-2})$

$(b_0, \ldots, b_{2n-2}) \xrightarrow{DFT} (d_0, \ldots, d_{2n-2})$.

$(c_0 d_0, \ldots, c_{2n-2} d_{2n-2}) \xrightarrow{IDFT} (e_0, e_1, \ldots, e_{2n-2})$

Thm: $(e_0, \ldots, e_{2n-2})$ is the convolution seq. of $(a_0, \ldots, a_{n-1})$ and $(b_0, \ldots, b_{n-1})$.

Running time: $O(n \log n)$.

($\ell = 2n-1$)

Pf:

$$e_k = \frac{1}{\ell} \sum_{j=0}^{\ell-1} c_j d_j \, \omega^{-k \cdot j}$$

$$= \frac{1}{\ell} \sum_{j=0}^{\ell-1} \left(\sum_{i=0}^{\ell-1} a_i \, \omega^{ij} \right) \left(\sum_{r=0}^{\ell-1} b_r \omega^{rj} \right) \omega^{-kj}$$

$$= \frac{1}{\ell} \sum_{i=0}^{\ell-1} \sum_{r=0}^{\ell-1} a_i b_r \sum_{j=0}^{\ell-1} \omega^{(i+r-k)\cdot j}$$

$$= \frac{1}{\ell} \sum_{i=0}^{\ell-1} \sum_{r=0}^{\ell-1} a_i b_r \left(\delta_{i+r,k} \cdot \ell \right)$$

← using claim from previous class.

$$= \sum_{(i,r) : i+r=k} a_i b_r .$$

□

Polynomial multiplication.

$$p(x) = \sum_{i=0}^{n-1} a_i x^i \; ; \; q(x) = \sum_{j=0}^{n-1} b_j x^j$$

$$e(x) = \sum_{l=0}^{2n-2} e_l x^l \; ; \; e(x) = p(x) q(x).$$

$$e_l = \sum_{i+j=l} a_i b_j .$$

Algorithm: Compute convolution of $(a_0, \ldots, a_{n-1})$ and $(b_0, \ldots, b_{n-1})$.

$O(n \log n)$ time algorithm.

Integer Multiplication:

$a = (a_0, a_1, \ldots, a_{n-1})$ ← bit string.

$b = (b_0, \ldots, b_{n-1})$

a_i's and b_j's are bits.

$$a = \sum_{i=0}^{n-1} a_i 2^i \quad ; \quad b = \sum_{j=0}^{n-1} b_j 2^j .$$

Define:

$$a(x) = \sum_{i=0}^{n-1} a_i x^i \qquad b(x) = \sum_{i=0}^{n-1} b_i x^i .$$

Compute: $c(x) = a(x)\, b(x)$ using above alg.

in $O(n \log n)$ time.

Evaluate $c(x)$ at $x = 2$.

Define: $c = c(2)$.

Claim: $c = a \cdot b$.