day 24 4/13/98 "You get me confused. You get me to say something and then I get nervous."

1 Fast Fourier Transforms (FFT)

1.1 motivation

Given a function there are several representations.

- polynomial: $f(x) = a_0 + a_1 x + ... a_{n-1} x^{n-1}$
- point set: $(0, f(0)), (1, f(1)), (2, f(2)), ...(n-1), f(n-1))$

The point set representation (with redundant sets) is used to transmit information in about a polynomial over a noisy channel. This message is decoded using a big matrix.

$$\begin{array}{ccccccccc}
1 & 0 & ... & ... & 0 & & a_0 & & f(0) \\
1 & 2 & 2^2 & ... & 2^{n-1} & & . & & f(2) \\
. & . & & & & * & . & = & . \\
. & . & & & & & . & & . \\
1 & cn & (cn)^2 & ... & (cn)^{n-1} & & a_{n-1} & & f(cn)
\end{array}$$

This is $O(n^3)$ to solve, or perhaps $O(n^{2.36})$ if you are clever.

There is a case (the FFT case) which is quickly solvable.

1.2 vector convolution

The convolution of two vectors $(a_0, ..., a_{n-1}) \otimes (b_0, ..., b_{n-1}) = (c_0, ..., c_{2n-2})$ is defined by $c_k = \sum_{i=0}^{k} a_i b_{k-i}$

1.3 algorithm

Compute $f(x)$ at $f(0), f(1), ..., f(2n-2)$ and $g(x)$ at $g(0), ..., g(2n-2)$. Then, we calculate $f(0)g(0), ..., f(2n-2)g(2n-2)$ to get $f(x)g(x) = c_0 + c_1 x + ... c_{2n-2} x^{2n-2}$. This doesn't look like it works except that by shifting to the complex plane, everything can be made cheap.

1.4 definitions

$\omega =$ primitive nth root of unity in the complex plane. In orther words $\omega = e^{i 2\pi \frac{j}{n}}$ where $j \neq n$.

Conditions:

- $\omega \neq 1$
- $\omega^p \neq 1$ with $1 \leq p \leq n-1$
- $\omega^n = 1$

- $\sum_{j=0}^{n-1} \omega^{jp} = 0$ for $1 \leq p \leq n-1$

The discrete fourier transform is defined as:

$$\begin{array}{ccccc} 1 & . & . & . & 1 \\ . & \omega & \omega^2 & \ldots & \\ . & \omega^2 & \omega^4 & \ldots & \\ . & & & & \\ 1 & . & . & . & \omega \end{array} * \begin{array}{c} a_0 \\ . \\ . \\ . \\ a_n \end{array} = \begin{array}{c} f(\omega^0) \\ . \\ . \\ . \\ f(\omega^{n-1}) \end{array}$$

We can express the matrix as DFT transform matrix as $F_{ij} = w^{ij}$

1.5 lemma

$(F^{-1})_{ij} = \frac{1}{n}\omega^{-ij}$

Proof:

multiply $(F^{-1})_{ij}F_{jk}$ using the property $w^n = 1$ and $\sum_{j=0}^{n-1} \omega^{jp} = 0$ for $1 \leq p \leq n-1$.

1.6 FFT

Let $D_{\frac{n}{2}} = \begin{array}{ccccc} 1 & & & & \\ & \omega & & & \\ & & \omega^2 & & \\ & & & \ddots & \\ & & & & -1 \end{array}$ Then $F_nV = \begin{array}{cc} F_{\frac{n}{2}} & D_{\frac{n}{2}}F_{\frac{n}{2}} \\ F_{\frac{n}{2}} & -D_{\frac{n}{2}}F_{\frac{n}{2}} \end{array} * \begin{array}{c} V_{even} \\ V_{odd} \end{array}$

This suggest the divide and conquer algorithm:

compute $F_{\frac{n}{2}}V_{even}$ and $D_{\frac{n}{2}}F_{\frac{n}{2}}V_{odd}$, then construct F_nV and return.

1.6.1 timing

$T(n) = 2 * T(\frac{n}{2}) + \frac{3}{2}n$ which means $T(n) = O(n \log n)$

1.7 FFT in parallel

The FFT algorithm is parallelizable. Arrange the nodes of the parallel processor on a hypercube. Each node has a particular value associated with it. Have the node look at its first (bitwise ordering) neighbor. These 2 nodes will have values a and b. Give one node $a + b$ and the other $a - b$. Repeat for each bit.

2 Shuffle exchange graph

let $V = \{(a_0, ..., a_n)|a_i = \{0, 1\}\}$

then the 'shuffle' takes $(a_0, ..., a_n) \rightarrow (a_1, ..., a_na_0)$

and the 'exchange' takes $(a_0, ..., a_n) = (ba_1, ..., a_n)$