Lecture 22: Fast Fourier Transform I

Convolutions

Def: Let $\{a_j\}_{j=0}^{n}$ and $\{b_k\}_{k=0}^{n}$ be two sequences of real numbers.
We define their convolution $a*b$ as the new sequence $\{c_\ell\}_{\ell=0}^{n+m}$ given by:

$$c_\ell = (a*b)_\ell = \sum_{j=0}^{\ell} a_j b_{\ell-j}$$

Convolutions appear naturally, such as in dice rolls or sums of PDFs.

We can also consider convolutions in a matrix view, where we sum the diagonals.

Lemma: Let $p(x) = \sum_{j=0}^{n} a_j x^j$ and $q(x) = \sum_{k=0}^{m} b_k x^k$ be two polynomials. Then the product $r(x) = p(x) \cdot q(x)$ has coefficients

$$r(x) = \sum_{\ell=0}^{m+n} c_\ell x^\ell$$

where $c_\ell = (a * b)_\ell$.

Proof: Expand the product.

Convolutions are also used in image processing for blurring, compressing, and feature extraction, among other uses.

Brute Force Algorithm: Use the direct formula, using running time $O(nm)$

HOWEVER! Fast Fourier Transform gives $O((n+m)\log(n+m))$ algorithm.

String Matching with Convolutions

Application: Fix $s = (s_0, \dots, s_n) \in \{0,1\}^{n+1}$, and let $p = (p_0, \dots, p_m) \in \{0,1\}^{m+1}$ be a pattern. WLOG $m \leq n$. Compute set of bits:

$$J := \{ \ell : s_{\ell+i} = p_i, \ \forall i = 0, \dots, m \}$$

Brute Force Alg: $O(m(n-m+1))$.

Idea: Use convolution to get $O((m+n)\log(m+n))$-time algorithm.

We should convolve reversed(p) with s to match the pattern. To check if the pattern matches, switch out 0 with -1 to simulate agreement of bits (xor).

Lemma: Define

$$a_j = \begin{cases} +1 & \text{if } s_j = 1 \\ -1 & \text{if } s_j = 0 \end{cases} \quad \text{and} \quad b_k = \begin{cases} +1 & \text{if } p_{m-k} = 1 \\ -1 & \text{if } p_{m-k} = 0 \end{cases}$$

Then $(a*b)_\ell = m+1$ iff $\ell - m \in \mathcal{H}$, and there is an $O(n \log n)$ time algorithm for $\mathcal{H}$.

Ex: Suppose $s = 011011$, and $p = 011$. Here, $n = 5$, $m = 2$. Then $\mathcal{H} = \{0, 3\}$.

Claim: $\forall\, m \leq \ell \leq n-m+1$,

$$(a*b)_\ell = m+1 - 2 \cdot \#\{\text{disagreements between } s_{\ell-m} \cdots s_\ell \text{ \& } p_1 \cdots p_m\}$$

Proof: $(a*b)_\ell = \sum_{j=\ell-m}^{\ell} a_j b_{\ell-j}$

$$= \sum_{j=\ell-m}^{\ell} \left(1 - 2\mathbb{1}[a_j \neq b_{l-j}]\right)$$

$$= (m+1) - 2\sum_{k=0}^{m} \mathbb{1}[s_{(\ell-m)+k} \neq p_k]$$

$= (m+1) -$ # of disagreements.

Convolutions in Geometry

Def: For two sets $A, B \subseteq \mathbb{R}^d$, define their Minkowski sum as the set

$$A+B := \{x+y : x \in A, y \in B\}$$

Lemma: For $A \subseteq \{0, \ldots, n\}$, $B \subseteq \{0, \ldots, m\}$, let $a \in \{0,1\}^{n+1}$ and $b \in \{0,1\}^{m+1}$ be indicator vectors. Then,

$$A+B = \{\ell : (a*b)_\ell > 0\}$$

Fast Fourier Transform

We use the polynomial version of convolutions. Note we can evaluate $r(x) = p(x) \cdot q(x)$ at any point in $O(n+m)$-time.

Fact: Any degree-n polynomial p is uniquely determined by its evaluations on any set of $n+1$ distinct points.

The Strategy

Input: Coefficient vectors

$\Big\Downarrow$

Compute evaluations

$x_j \mapsto p(x_j)$, $x_j \mapsto q(x_j)$

for $j = 0 \dots n+m$

Find good choice of $x_0 \dots x_{n+m} \in \mathbb{C}$ using DaC to get $O((n+m)\log(n+m))$ time

$\xRightarrow[\;O(n+m)\text{ time}\;]{}$

$\cdot$ #evaluations

evaluations

$\Big\Uparrow$ Interpolate back

Output: Coefficient vec $a*b$ of $r(x) = p(x)\cdot q(x)$

Idea: Let $x \in \mathbb{R}$ nonzero. Evaluate $p(x)$ & $p(-x)$ for the price of computing one of them (maybe up to add. $O(1)$).

$$p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots$$

$$p(-x) = a_0 - a_1 x + a_2 x^2 - a_3 x^3 + \cdots$$

Define:

$$\left.\begin{aligned} P_{even}(x) &= a_0 + a_2 x + a_4 x^2 + \cdots = \sum_{k=0}^{\frac{n}{2}-1} a_{2k} \cdot x^k \\ P_{odd}(x) &= a_1 + a_3 x + a_5 x^2 + \cdots = \sum_{k=0}^{\frac{n}{2}-1} a_{2k+1} x^k \end{aligned}\right\} \text{degree } \frac{n}{2}-1$$

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

$p(-x) = P_{even}(x^2) - x \cdot P_{odd}(x^2)$

Define $T(n) :=$ time it takes to evaluate degree $(n-1)$ poly on n inputs

$$T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \lg n)$$:)

Obs: If we evaluate p at $x_0, \ldots x_{n-1}$, then we need to evaluate P_{even} & P_{odd} at $x_0^2, \ldots x_{n-1}^2$. Want $x_0, \ldots x_{n-1}$ to come in $\pm$ pairs.

Let S_k = {points on which we evaluate a polynomial of degree $(2^k - 1)$}, $|S_k| = 2^k$

Recurrence: $= \{ \sqrt{x}, -\sqrt{x} : x \in S_k \}$ Base Case: $S_0 = \{1\}$

$$= \{ y \in \mathbb{C} : y^2 \in S_k \}$$

$$S_0 = \{1\}$$

$$S_1 = \{x : x^2 = 1\} = \{-1, +1\}$$

$$S_2 = \{x : x^4 = 1\} = \{\pm 1, \pm i\}$$

$$\vdots$$

$$S_K = \left\{ x \in \mathbb{C} \mid x^{2^K} - 1 = 0 \right\} = \left\{ x = e^{\frac{i\,2\pi}{2^K}} \ \forall i = 0 \ldots 2^K - 1 \right\}$$

Discrete Fourier Transform: Given $\{a_j\}_{j=0}^{n-1}$ specifying a poly $p(x)=\sum_{j=0}^{n-1} a_j x^j$, complete all the evaluations

$$\omega_n^k \longmapsto p(\omega_n^k) \quad \forall \; k=0,\ldots,n-1$$

where ω_n is the primitive n^{th} root of unity.

FFT for computing DFT:

Input: $a_0, \dots a_{n-1}$; Assume n is power of 2

If $n = 1$:
  Return $[a_0]$
Build $a_{even} := (a_0, a_2 \dots a_{n-2})$, $a_{odd} := (a_1, a_3 \dots a_{n-1})$ $\Big\}\ O(n)$

Recurse $F_{even} = FFT(a_{even})$, $F_{odd} = FFT(a_{odd})$ $\Big\}\ 2T(n/2)$

For $k \in \{0, 1, \dots n/2 - 1\}$:
  Set $F[k] := F_{even}[k] + \omega_n^k \cdot F_{odd}[k]$
  Set $F[k + \frac{n}{2}] = F_{even}[k] + \omega_n^{k + n/2} \cdot F_{odd}[k]$ $\Big\}\ O(n)$

Return F

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