Tutorial 11

Gidon Rosalki

2025-01-23

1 FFT

A polynomial of power $n-1$:

$$\begin{aligned} p(x) &= \sum_{i=0}^{n-1} a_i x^i \\ &= \left\langle \begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \end{bmatrix}, \begin{bmatrix} x^0 \\ \vdots \\ x^{n-1} \end{bmatrix} \right\rangle \end{aligned}$$

which is called the coefficient representation of the polynomial.
Value representation: We will say that $\{(x_i, y_i)\}_{i=0}^{n-1}$ is the value representation of the polynomial $\forall i = 0 \dots n-1 : \ p(x_i) = y_i$.

Theorem 1 (Lagrangian algorithm)**.** *For every set of n points $\{(x_i, y_i)\}_{i=0}^{n-1}$ where all the xs are different, then there exists a single polynomial of power $\leq n-1$ such that $p(x_i) = y_i$ for every i.*

Proof. (Not exactly a proof, not written as such elsewhere)
Let there be the value representing vector

$$v = (p(x_0), \dots, p(x_{n-1}))$$

So

$$v = V \begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

Where V is the Vandermonde matrix, which is invertible **if and only if** $\forall i \neq j, \ x_i \neq x_j$. Since the Vandermonde matrix is of size $n \times n$, the naive method to change representation is $O(n^2)$, but perhaps we can do this in $O(n \log(n))$?
Multiplying polynomials: Let there be $p(x), q(x)$, we are given $x_0, \dots, x_{n-1}$ at the beginning.

$$p(x_0) q(x_0) = p \cdot q(x_0) \implies O(n)$$

$$\begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ \vdots \\ b_{n-1} \end{bmatrix} = O(n^2)$$

so, we would much rather calculate multiplication from the coefficient representation than the value representation, but transferring between the representations through multiplying by Vandermonde takes $O(n^2)$. □

2 Convolution

Definition 2.1 (Convolution (linear))**.** *Given $a = (a_0, \dots, a_{n-1})$, $b = (b_0, \dots, b_{m-1})$, then let $c = a * b = (c_0, \dots, c_{n+m-2})$, where*

$$c_i = \sum_{j=0}^{i} a_j b_{i-j}$$

and if an index does not exist, it is defined to be 0.

Example 1.

$$a = (a_0, a_1, a_2)$$
$$b = (b_0, b_1)$$
$$c_0 = a_0 b_0$$
$$c_1 = a_0 b_1 + b_0 a_1$$
$$c_2 = a_1 b_1 + b_0 a_2 + (b_2 a_0 = 0)$$
$$c_3 = b_1 a_2 + (a_1 b_2 = 0)$$

Consider:

$$\begin{aligned}\left(b_0 x^0 + b_1 x^1\right)\left(a_0 x^0 + a_1 x^1 + a_2 x^2\right) &= b_0 a_0 x^0 + (a_0 b_1 + a_1 b_0)\, x^1 + (b_1 a_1 + b_0 a_2)\, x^2 + (b_1 a_2)\, x^3 \\ &= c_0 x^0 + c_1 x^1 + c_2 x^2 + c_3 x^3\end{aligned}$$

So we can see that multiplying polynomials is simply an instance of convolution!

2.1 Uses of convolution

2.1.1 Pattern matching

Input: 2 strings $s \in \{\pm 1\}^n,\ p \in \{\pm 1\}^m$, we will assume without loss of generality $m \leq n$
Output: Every index in s where they start a continuous series of p

Example:

$$s = (1, -1, 1, 1, -1, 1)$$
$$p = (1, -1, 1)$$
$$D = \{0, 3\}$$

Naive solution: We may solve this by passing over all of s, and checking if p appears: $O(nm)$

We will see a more efficient solution with FFT/convolution:

$$s_0 p_0 + s_1 p_1 + s_2 p_2 = m$$

So we know that there is an instance of p at index 0.

$$s_3 p_0 + s_4 p_1 + s_5 p_2 = m$$

This also holds true, but how do we get here from convolution? Let us consider

$$s = (1, -1, 1)$$
$$p = (a, b, c)$$
$$(s * p)_0 = 1 \cdot a$$
$$(s * p)_1 = 1 \cdot b + (-1) \cdot a$$
$$(s * p)_2 = 1 \cdot a + 1 \cdot c + (-1) \cdot b$$

Notice that at $(s * p)_2$ that is the same as performing $s_0 p_0, \ldots$ on the **inverse** of p.
Efficient algorithm:

1. We will find p^R where $\forall i = 0, \ldots, m-1\ p_i^R = p_{m-1-i}$: $O(m)$
2. We will find the convolution of p^R with s, and call it c: We will use a fast algorithm for multiplying polynomials based off FFT in $O(n \log(n))$
3. We will define and return

$$D = \{k : c_{k+m-1} = m\}$$

$O(n + m)$

Runtime: $O(m) + O(n\log(n)) + O(n+m) = O(n\log n)$ Correctness:

Theorem 2. *$C_{k+m-1} = m \Leftrightarrow$ there exists a continuous p at $s_k \Leftrightarrow s_k s_{k+1} \ldots s_{k+m-1} = p \Leftrightarrow \forall i = 0 \ldots m-1\ s_{k+i} = p_i$*

Proof .

$$
\begin{aligned}
c_i &\overset{def}{=} \sum_{j=0}^{i} s_j p^R_{i-j} \\
\implies c_{k+m-1} &\overset{def}{=} \sum_{j=0}^{k+m-1} s_j p^R_{k+m-1-j} \\
\text{Definition of } p^R \quad &= \sum_{j=0}^{k+m-1} s_j p_{j-k} \\
&= \sum_{j=0}^{k} s_j p_{j-k} + \sum_{j=k}^{k+m-1} s_j p_{j-k} \\
(j \le k-1 \implies j-k<0 \implies p_{j-k}=0) &= 0 + \sum_{j=k}^{k+m-1} s_j p_{j-k} \\
(i = j-k\ j = i+k) &= \sum_{i=0}^{m-1} s_{i+k} p_i \\
\implies c_{k+m-1} &= \sum_{i=0}^{m-1} s_{i+k} p_i
\end{aligned}
$$

So our theorem is now

$$\sum_{i=0}^{m-1} s+k+ip_i = m \Leftrightarrow ...$$

$s_{ki}p_i \in \{1,-1,0\}$, from the definition of s,p convolution. The sum over m variables therefore

$$\sum_{i=0}^{m-1} s_{k+i} p_i \le m$$

with equality when all the variables are 1, which happens from the definitions of s,p when for all $i = 0,\ldots,m-1$ $s_{k+i} = p_i \neq 0$, which is the definition of the continuous appearance of p at the start s_k. $\square$

3 DFT - Discrete Fourier Transform

Given a polynomial with the coefficient vector $\begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \end{bmatrix}$, then

$$DFT_n\left(\begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \end{bmatrix}\right) = \begin{bmatrix} p\left(\omega_n^0\right) \\ \vdots \\ p\left(\omega_n^{n-1}\right) \end{bmatrix}$$

which is a representation of the nth roots of unity.

$$DFT_n\left(\begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \end{bmatrix}\right) = F\begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} \left(\omega_n^0\right)^0 & \ldots & \left(\omega_n^0\right)^{n-1} \\ \vdots & \ddots & \vdots \\ \left(\omega_n^{n-1}\right)^0 & \ldots & \left(\omega_n^{n-1}\right)^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

where

$$F_{ij} = \left(\omega_n^i\right)^j$$

4 FFT - Fast Fourier Transform

This is the algorithm for finding DFT in $O(n\log(n))$.

4.1 Polynomial multiplication

Input: $p = (a_0, \ldots, a_{n-1}),\ q = (b_0, \ldots, b_{n-1})$
Output: $qp = (c_0, \ldots, c_{2n-2})$
Algorithm:

1. We will bolster with 0s q, p to the smallest power of 2 which is greater than or equal to $2n-1$, and call them $\overline{p}, \overline{q} \in \mathbb{C}^{2^k} : 2n-1 \leq 2^k$
2. We will calculate $FFT(\overline{p}),\ FFT(\overline{q})$, and we have got the value representations
3. We will multiply value value, and get the value representation of $\overline{pq}$
4. We will return the coefficient representation with FFT^{-1}:

$$q \cdot p = FFT^{-1}\left(FFT(\overline{p}) \cdot FFT(\overline{q})\right)$$

Runtime:

1. Bolstering: $O(n)$
2. FFT: $O(n\log(n))$
3. Value value multiplication: $O(n)$
4. Inverse FFT: $O(n\log(n))$

Total: $O(n\log(n))$