CS 218 Design and Analysis of Algorithms

Nutan Limaye

Indian Institute of Technology, Bombay
nutan@cse.iitb.ac.in

Module 1: Basics of algorithms

Divide, Delegate and Combine (Divide and Conquer)

You cannot do everything and be efficient!

Univariate polynomial multiplication

Input: Two univariate polynomials $A(x) = a_0 + a_1x + \ldots, a_{n-1}x^{n-1}$, $B(x) = b_0 + b_1x + \ldots, b_{n-1}x^{n-1}$ described by the coefficient vectors of length n.

Output: $C(x) = A(x) \times B(x)$ as a coefficient vector of length $2n - 1$

How many + or × operations are enough to compute it?

We can easily see that

$$c_k = \sum_{i,j<n, i+j=k} a_i b_j$$

For two given vectors $a = (a_0, \ldots, a_{n-1})$ and $b = (b_0, \ldots, b_{n-1})$, $c = (c_0, \ldots, c_{2n-2})$ is called the convolution of these vectors.

As seen here, convolution occurs naturally in polynomial multiplication.

It shows that $O(n^2)$ algorithm is easy.

Univariate polynomial multiplication

Input: Two univariate polynomials $A(x) = a_0 + a_1x + \dots, a_{n-1}x^{n-1}$, $B(x) = b_0 + b_1x + \dots, b_{n-1}x^{n-1}$ described by the coefficient vectors of length n.

Output: $C(x) = A(x) \times B(x)$ as a coefficient vector of length $2n - 1$

How many $+$ or $\times$ operations are enough to compute it?

We can easily see that

$$c_k = \sum_{i,j<n, i+j=k} a_i b_j$$

It shows that $O(n^2)$ algorithm is easy.

Can we do better?

Can we get $O(n \log n)$ time algorithm?

Univariate polynomial multiplication

What is a way to do this?

Rather than multiplying A and B symbolically, consider evaluations of these polynomials.

Think of A, B as functions of x and do as follows:

(a) Choose $2n$ values $\alpha_1, \ldots, \alpha_{2n}$ and evaluate $A(x)$ and $B(x)$ on all these points.

(b) Now $C(\alpha_j) = A(\alpha_j) \cdot B(\alpha_j)$ for each $j \in [2n-1]$.

(c) Recover $C(x)$, i.e the entire vector $(c_0, \ldots, c_{2n-2})$, from the evaluations $C(\alpha_1), \ldots, C(\alpha_{2n-1})$.

Univariate polynomial multiplication

The features about the approach

The step (b) takes only $O(n)$ operations.

That is a positive thing.

Evaluating a polynomial on a single value will take $\Omega(n)$ operations.

And we need $O(n)$ such evaluations.

Thus giving the running time of $O(n^2)$. This seems problematic.

- Divide and conquer
- Reuse a computation

Polynomial evaluation

Can we use divide and conquer idea for polynomial evaluation?

Let us define two polynomials. $A_0(x)$ and $A_1(x)$ as follows.

$$A_0(x) = a_0 + a_2 x + a_4 x^2 + \ldots + a_{n-2} x^{n-2/2}$$

and

$$A_1(x) = a_1 + a_3 x + a_5 x^2 + \ldots + a_{n-1} x^{n-2/2}$$

It is not very difficult to see that

$$A(x) = A_0(x^2) + x A_1(x^2). \tag{1}$$

Here the degree of $A_0(x)$ and $A_1(x)$ is half of the degree of $A(x)$.

Using $A_0(x)$ and $A_1(x)$, we can compute $A(x)$ in $O(1)$ operations.

Where should we evaluate the polynomials?

Main Idea behind getting gains.

The $2n$ values where we wish to perform evaluation should be intimately related to each other.

This is to help evaluations of A and B on all these points to become shareable across many evaluations.

Let $\omega_{j,k} = e^{(2\pi i)j/k}$ for $j \in \{0, \ldots, k-1\}$, be the roots of the equation $x^k = 1$ over the complex numbers $\mathbb{C}$.

They are also called the k^{th} roots of unity.

For our $2n$ evaluations, we will use $2n^{\text{th}}$ roots of unity.

Polynomial evaluation

How do we gain by this?

First note that $\omega_{j,2n}^2 = (e^{(2\pi i)j/2n})^2 = e^{2\pi ji/n} = \omega_{j,n}$.

Hence if we square $\omega_{j,2n}$ for any $j \in \{0, \dots, 2n-1\}$, then we get $\omega_{j,n}$.

Now, let us reconsider Equation 1. $A(x) = A_0(x^2) + xA_1(x^2)$.

[illegible]

$$\begin{aligned} A(\omega_{j,2n}) &= A_0((\omega_{j,2n})^2) + \omega_{j,2n} \cdot A_1((\omega_{j,2n})^2) \\ &= A_0(\omega_{j,n}) + \omega_{j,2n} \cdot A_1(\omega_{j,n}) \end{aligned}$$

Polynomial evaluation

How do we gain by this?

$$\begin{aligned} A(\omega_{j,2n}) &= A_0((\omega_{j,2n})^2) + \omega_{j,2n} \cdot A_1((\omega_{j,2n})^2) \\ &= A_0(\omega_{j,n}) + \omega_{j,2n} \cdot A_1(\omega_{j,n}) \end{aligned}$$

Now note that $A_0(\omega_{j,n})$ and $A_1(\omega_{j,n})$ are evaluations of $A_0(x)$ and $A_1(x)$ at the n^{th} roots of unity.

The polynomials $A_0(x)$ and $A_1(x)$ are of degree $(n-2)/2$.

Therefore we can assume that recursively we have these evaluations!

Now as noted before, we only need $O(1)$ to combine them.

Polynomial evaluation

Overall time for polynomial evaluations.

To evaluate $A(x)$ at $2n^{\text{th}}$ roots of unity, we invoke a recursive procedure using $A(x) = A_0(x^2) + xA_1(x^2)$ this identity.

Degrees of $A_0(x)$ and $A_1(x)$ are around half of degree of $A(x)$.

Recursively we have evaluations of $A_0(x), A_1(x)$ at n^{th} roots of unity.
To find evaluations of $A(x)$ at $\omega_{j,2n}$ for each $j \in \{0, \ldots, 2n-1\}$ using these takes $O(1)$ time.

$T(2n-1) = 2T((n-2)/2) + 2n - 1 = O(n \log n)$.

Univariate polynomial multiplication

Recall our approach.

> Rather than multiplying A and B symbolically, consider evaluations of these polynomials.

(a) Choose $2n$ values $\alpha_1, \ldots, \alpha_{2n}$ and evaluate $A(x)$ and $B(x)$ on all these points.

(b) Now $C(\alpha_j) = A(\alpha_j) \cdot B(\alpha_j)$ for each $j \in [2n-1]$.

(c) Recover $C(x)$, i.e the entire vector $(c_0, \ldots, c_{2n-1})$, from the evaluations $C(\alpha_1), \ldots, C(\alpha_{2n-1})$.

Can interpolation be done in time $O(n \log n)$?

Univariate polynomial interpolation

The standard approach

For any univariate polynomial of degree $2n-1$, if we have evaluations of it at $2n$ points, then we can recover the polynomial.

Typically, the interpolation done at arbitrary points can take $O(n^3)$ operations, as it involves computing a certain determinant.

Can we do anything better?

Univariate polynomial interpolation

Another interesting idea.

Instead of doing the standard interpolation, use the fact that we have evaluations at special points.

In fact, we will reduce the problem of interpolation of $C(x)$ to the evaluation of another polynomial, say $D(x)$.

The reduction will be a re-interpretation of $C(x)$.

As we know evaluations can be done in time $O(n \log n)$, we will be done.

Interpolation to evaluation

How do we do this reduction?

Let $C(x) = \sum_{s=0}^{2n-1} c_s \cdot x^s$.

Let us define a new polynomial $D(x) = \sum_{s=0}^{2n-1} d_s x^s$, where $d_s = C(\omega_{s,2n})$.

We will prove the following lemma.

Lemma

For any polynomial $C(x) = \sum_{s=0}^{2n-1} c_s \cdot x^s$ *and the corresponding polynomial* $D(x) = \sum_{s=0}^{2n-1} C(\omega_{s,2n}) x^s$, *then we have* $c_s = \frac{1}{2n} D(\omega_{2n-s,2n})$

See Kleinberg and Tardos for the proof of this lemma.

Interpolation to evaluation

Lemma

For any polynomial $C(x) = \sum_{s=0}^{2n-1} c_s \cdot x^s$ *and the corresponding polynomial* $D(x) = \sum_{s=0}^{2n-1} C(\omega_{s,2n})x^s$*, then we have* $c_s = \frac{1}{2n} D(\omega_{2n-s,2n})$

Now that we have the lemma, we have shown that even step (c) can be done in time $O(n \log n)$.

So overall, Step (a) and (c) take $O(n \log n)$ each.

Step (b) takes $O(n)$ time.

We are done! This approach is called Fast Fourier Transform.

Polynomial multiplication

(a) Choose $2n$ values $\alpha_1, \dots, \alpha_{2n}$ and evaluate $A(x)$ and $B(x)$ on all these points.

(b) Now $C(\alpha_j) = A(\alpha_j) \cdot B(\alpha_j)$ for each $j \in [2n-1]$.

(c) Recover $C(x)$, i.e the entire vector $(c_0, \dots, c_{2n-1})$, from the evaluations $C(\alpha_1), \dots, C(\alpha_{2n-1})$.

All steps put together, the task is finally accomplished!