Math 788: Computational Number Theory

- Math 788 Computational Number Theory Syllabus
- Math 788 Computational Number Theory Course Description
- Math 788 Computational Number Theory Class Notes
- Lectures on Primality Testing in Polynomial Time
- Computational Material on Polynomials
- Lectures

Lecture 1	Lecture 2	Lecture 3	Lecture 4	Lecture 5
Lecture 6	Lecture 7	Lecture 8	Lecture 9	Lecture 10
Lecture 11	Lecture 12	Lecture 13	Lecture 14	Lecture 15
Lecture 16	Lecture 17	Lecture 18	Lecture 19	Lecture 20
Lecture 21	Lecture 22	Lecture 23	Lecture 24	Lecture 25
Lecture 26				

- Test Materials

  Review List
  Test from 2007
  Final Exam from 2007
  Old Comp Exam Problems

- Graduate Number Theory Courses At The University of South Carolina

- Lectures on Primality Testing in Polynomial Time
- Computational Material on Polynomials
- **Lectures**

Lecture 1	Lecture 2	Lecture 3	Lecture 4	Lecture 5
Lecture 6	Lecture 7	Lecture 8	Lecture 9	Lecture 10
Lecture 11	Lecture 12	Lecture 13	Lecture 14	Lecture 15
Lecture 16	Lecture 17	Lecture 18	Lecture 19	Lecture 20
Lecture 21	Lecture 22	Lecture 23	Lecture 24	Lecture 25
Lecture 26				

- **Test Materials**

  Review List
  Test from 2007
  Final Exam from 2007
  Old Comp Exam Problems

- Graduate Number Theory Courses At The University of South Carolina

1. Be able to do problems related to any of the ~~homework~~. **Practice Problems**

3. Be able to define a strong pseudoprime to the base b and be able to prove that no n is a strong pseudoprime to every base b with $1 \le b \le n$ and $\gcd(b, n) = 1$.

2. Be able to prove that $\gcd(u, v)$ is $\asymp \log N$ on average and that usually it's much smaller.

5. Be able to state and prove the Proth, Pocklington, Lehmer Test for primality.

6. Be able to prove that most numbers n have a prime factor $> \sqrt{n}$.

8. Be able to factor n using Dixon's Algorithm (see homework).

8. Be able to factor n using Dixon's Algorithm (see homework). (You will be given some

9. Be able to factor n using the Quadratic Sieve Algorithm.

tion and you will need to make use of it and give the remaining details.)

10. Be able to prove Landau's inequality for the size of the factors of a polynomial.

12. Be able to state and prove Hadamard's inequality.

12. Be able to state and prove Hadamard's inequality. (You do not need to be able to define the

13. Be able to define what it means for a basis in a lattice to be *reduced*.

14. Be able to show (10) in the notes, that is that

14. Be able to prove $\mathbf{b} \in \mathcal{L},\ \mathbf{b} \neq \mathbf{0} \implies \|\mathbf{b}_1\| \le 2^{(n-1)/2}\|\mathbf{b}\|$.

You have the power to request and even make changes to the list above. I have the power to veto the changes.

Discrete and Fast Fourier Transforms

Goal: Let $M(d)$ denote the number of binary bit operations needed to multiply two positive integers each with $\le d$ bits. We stated that $M(d) \ll d(\log d)\log\log d$. We give the basic idea behind the proof, not worrying so much about the running time but concentrating on the main idea of using fast Fourier transforms for performing multiplication.

Lemma. *Let n and k be integers with $n \ge 1$. Let $\omega = e^{2\pi i/n}$. Then*

$$\sum_{j=0}^{n-1} \omega^{kj} = \begin{cases} 0 & \text{if } k \not\equiv 0 \pmod{n} \\ n & \text{if } k \equiv 0 \pmod{n}. \end{cases}$$

Example. $1 - \frac{1}{4} + \frac{1}{7} - \frac{1}{10} + \cdots = \frac{3\log 2 + \sqrt{3}\pi}{9}$

Lemma. *Let n and k be integers with $n \geq 1$. Let $\omega = e^{2\pi i/n}$. Then*

$$\sum_{j=0}^{n-1} \omega^{kj} = \begin{cases} 0 & \text{if } k \not\equiv 0 \pmod{n} \\ n & \text{if } k \equiv 0 \pmod{n}. \end{cases}$$

Definitions and notations. For n a positive integer, we set $\omega = \omega_n = e^{2\pi i/n}$. Let

$$D = D(n, \omega) = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)^2} \end{pmatrix}.$$

For $\vec{u} = \langle u_0, u_1, \ldots, u_{n-1} \rangle \in \mathbb{C}^n$, define $\vec{v} = \langle v_0, v_1, \ldots, v_{n-1} \rangle$, called the discrete Fourier transform of $\vec{u}$, by $\vec{v} = D\,\vec{u}^T$. The inverse discrete Fourier transform of a vector $\vec{v} \in \mathbb{C}^n$ is defined as $(1/n)D(n, \omega^{-1})\,\vec{v}^T$.

Why is $(1/n)D(n, \omega^{-1})D(n, \omega)$ the identity matrix?

A Polynomial Connection

Observe that if $f(x) = \sum_{j=1}^{n-1} a_j x^j \in \mathbb{C}[x]$, then

$$D\,\langle a_0, a_1, \ldots, a_{n-1}\rangle^T = \langle f(1), f(\omega), \ldots, f(\omega^{n-1})\rangle^T.$$

On the other hand, if we know $f(1), f(\omega), \ldots, f(\omega^{n-1})$ and do not know the coefficients of $f(x)$, then we can obtain the coefficients from

$$D^{-1}\langle f(1), f(\omega), \ldots, f(\omega^{n-1})\rangle^T = \langle a_0, a_1, \ldots, a_{n-1}\rangle^T.$$

Note that here $D^{-1} = (1/n)D(n, \omega^{-1})$.

If

$$F(x) = f(1) + f(\omega)x + \cdots + f(w^{n-1})x^{n-1},$$

then

$$F(1) = na_0,\ F\left(\omega^{-1}\right) = na_1,\ \ldots, F\left(\omega^{-(n-1)}\right) = na_{n-1}.$$

The Fast Fourier Transform

We explain a fast way of performing the computation in

$$(*) \qquad D\,\langle a_0, a_1, \ldots, a_{n-1}\rangle^T = \langle f(1), f(\omega), \ldots, f(\omega^{n-1})\rangle^T.$$

There exist unique polynomials f_e and f_o in $\mathbb{C}[x]$ such that $f(x) = f_e(x^2) + x f_o(x^2)$. Calculate $f_e(\omega^{2j})$, $f_o(\omega^{2j})$ and $\omega^j f_o(\omega^{2j})$ for $0 \le j \le n-1$ to obtain the right side of $(*)$.

Why is this Fast?

For convenience, view n as a power of 2. Let $A(n)$ be the number of arithmetic operations that one needs to compute $f(1), f(\omega), \ldots, f(\omega^{n-1})$. Computing in an obvious way gives $A(n) \le n^2 + n(n-1)$. Observe that

$$f_e(\omega_n^{2j}) = f_e(\omega_n^{2(j+(n/2))}) \qquad \text{for } 0 \le j \le (n/2) - 1.$$

$$(*) \qquad D\,\langle a_0, a_1, \ldots, a_{n-1}\rangle^T = \langle f(1), f(\omega), \ldots, f(\omega^{n-1})\rangle^T.$$

There exist unique polynomials f_e and f_o in $\mathbb{C}[x]$ such that $f(x) = f_e(x^2) + x f_o(x^2)$. Calculate $f_e(\omega^{2j})$, $f_o(\omega^{2j})$ and $\omega^j f_o(\omega^{2j})$ for $0 \le j \le n-1$ to obtain the right side of $(*)$.

For convenience, view n as a power of 2. Let $A(n)$ be the number of arithmetic operations that one needs to compute $f(1), f(\omega), \ldots, f(\omega^{n-1})$. Computing in an obvious way gives $A(n) \le n^2 + n(n-1)$. Observe that

$$f_e(\omega_n^{2j}) = f_e(\omega_n^{2(j+(n/2))}) \qquad \text{for } 0 \le j \le (n/2) - 1.$$

Similar equations hold with f_e replaced by f_o. We deduce that computing $f_e(\omega^{2j})$ and $f_o(\omega^{2j})$ takes $2A(n/2)$ arithmetic operations. Multiplying $f_o(\omega^{2j})$ by ω^j and then adding $f_e(\omega^{2j})$ takes $2n$ more arithmetic operations. We obtain

$$A(n) = 2A(n/2) + 2n \quad \overset{?}{\underset{?}{\Longrightarrow}} \quad A(n) = (2/\log 2)\, n \log n + Cn$$

for some constant C.

The Fast Fourier Transform

We explain a fast way of performing the computation in

$$(*) \qquad D\,\langle a_0, a_1, \ldots, a_{n-1}\rangle^T = \langle f(1), f(\omega), \ldots, f(\omega^{n-1})\rangle^T.$$

There exist unique polynomials f_e and f_o in $\mathbb{C}[x]$ such that $f(x) = f_e(x^2) + x f_o(x^2)$. Calculate $f_e(\omega^{2j})$, $f_o(\omega^{2j})$ and $\omega^j f_o(\omega^{2j})$ for $0 \leq j \leq n-1$ to obtain the right side of $(*)$.

Why is this Fast?

For convenience, view n as a power of 2. Let $A(n)$ be the number of arithmetic operations that one needs to compute $f(1), f(\omega), \ldots, f(\omega^{n-1})$. Computing in an obvious way gives $A(n) \leq n^2 + n(n-1)$. Observe that

$$f_e(\omega_n^{2j}) = f_e(\omega_n^{2(j+(n/2))}) \qquad \text{for } 0 \leq j \leq (n/2) - 1.$$

$$(*)\qquad D\,\langle a_0, a_1, \ldots, a_{n-1}\rangle^T = \langle f(1), f(\omega), \ldots, f(\omega^{n-1})\rangle^T.$$

There exist unique polynomials f_e and f_o in $\mathbb{C}[x]$ such that $f(x) = f_e(x^2) + x f_o(x^2)$. Calculate $f_e(\omega^{2j})$, $f_o(\omega^{2j})$ and $\omega^j f_o(\omega^{2j})$ for $0 \le j \le n-1$ to obtain the right side of $(*)$.

For convenience, view n as a power of 2. Let $A(n)$ be the number of arithmetic operations that one needs to compute $f(1), f(\omega), \ldots, f(\omega^{n-1})$. Computing in an obvious way gives $A(n) \le n^2 + n(n-1)$. Observe that

$$f_e(\omega_n^{2j}) = f_e(\omega_n^{2(j+(n/2))}) \qquad \text{for } 0 \le j \le (n/2) - 1.$$

Similar equations hold with f_e replaced by f_o. We deduce that computing $f_e(\omega^{2j})$ and $f_o(\omega^{2j})$ takes $2A(n/2)$ arithmetic operations. Multiplying $f_o(\omega^{2j})$ by ω^j and then adding $f_e(\omega^{2j})$ takes $2n$ more arithmetic operations. We obtain

$$A(n) = 2A(n/2) + 2n \implies A(n) = (2/\log 2)\, n \log n + Cn$$

for some constant C.

$$(*) \qquad D\,\langle a_0, a_1, \ldots, a_{n-1}\rangle^T = \langle f(1), f(\omega), \ldots, f(\omega^{n-1})\rangle^T$$

$$(**) \qquad D^{-1}\langle f(1), f(\omega), \ldots, f(\omega^{n-1})\rangle^T = \langle a_0, a_1, \ldots, a_{n-1}\rangle^T$$

Comment: Each of these can be computed using $O(n \log n)$ complex arithmetic operations.

Multiplying Two Polynomials

An obvious way of obtaining the coefficients of a product of two polynomials of degree $\leq n$ takes on the order of n^2 multiplications.

Multiplying Two Polynomials

An obvious way of obtaining the coefficients of a product of two polynomials of degree $\leq n$ takes on the order of n^2 multiplications. We can do better (but using arithmetic with nth roots of unity).

Multiplying Two Polynomials

An obvious way of obtaining the coefficients of a product of two polynomials of degree $\leq n$ takes on the order of n^2 multiplications. We can do better (but using arithmetic with nth roots of unity). Let

$$g(x) = b_{r-1}x^{r-1} + \cdots + b_0 \quad \text{and} \quad h(x) = c_{s-1}x^{s-1} + \cdots + c_0$$

be polynomials in $\mathbb{C}[x]$ with $s \leq r$.

Multiplying Two Polynomials

An obvious way of obtaining the coefficients of a product of two polynomials of degree $\leq n$ takes on the order of n^2 multiplications. We can do better (but using arithmetic with nth roots of unity). Let

$$g(x) = b_{r-1}x^{r-1} + \cdots + b_0 \quad \text{and} \quad h(x) = c_{s-1}x^{s-1} + \cdots + c_0$$

be polynomials in $\mathbb{C}[x]$ with $s \leq r$. Let $n = 2r$, and rewrite $g(x)$ and $h(x)$ with leading zeroes as

$$g(x) = b_{n-1}x^{n-1} + \cdots + b_0 \quad \text{and} \quad h(x) = c_{n-1}x^{n-1} + \cdots + c_0.$$

Multiplying Two Polynomials

An obvious way of obtaining the coefficients of a product of two polynomials of degree $\leq n$ takes on the order of n^2 multiplications. We can do better (but using arithmetic with nth roots of unity). Let

$$g(x) = b_{r-1}x^{r-1} + \cdots + b_0 \quad \text{and} \quad h(x) = c_{s-1}x^{s-1} + \cdots + c_0$$

be polynomials in $\mathbb{C}[x]$ with $s \leq r$. Let $n = 2r$, and rewrite $g(x)$ and $h(x)$ with leading zeroes as

$$g(x) = b_{n-1}x^{n-1} + \cdots + b_0 \quad \text{and} \quad h(x) = c_{n-1}x^{n-1} + \cdots + c_0.$$

Let $f(x) = g(x)h(x)$, and note that $\deg f \leq n - 1$.

$$(*) \qquad D\,\langle a_0, a_1, \ldots, a_{n-1}\rangle^T = \langle f(1), f(\omega), \ldots, f(\omega^{n-1})\rangle^T$$

An obvious way of obtaining the coefficients of a product of two polynomials of degree $\leq n$ takes on the order of n^2 multiplications. We can do better (but using arithmetic with nth roots of unity). Let

$$g(x) = b_{r-1}x^{r-1} + \cdots + b_0 \quad \text{and} \quad h(x) = c_{s-1}x^{s-1} + \cdots + c_0$$

be polynomials in $\mathbb{C}[x]$ with $s \leq r$. Let $n = 2r$, and rewrite $g(x)$ and $h(x)$ with leading zeroes as

$$g(x) = b_{n-1}x^{n-1} + \cdots + b_0 \quad \text{and} \quad h(x) = c_{n-1}x^{n-1} + \cdots + c_0.$$

Let $f(x) = g(x)h(x)$, and note that $\deg f \leq n - 1$.

$$(*) \qquad D\,\langle a_0, a_1, \ldots, a_{n-1}\rangle^T = \langle f(1), f(\omega), \ldots, f(\omega^{n-1})\rangle^T$$

An obvious way of obtaining the coefficients of a product of two polynomials of degree $\leq n$ takes on the order of n^2 multiplications. We can do better (but using arithmetic with nth roots of unity). Let

$$g(x) = b_{r-1}x^{r-1} + \cdots + b_0 \quad \text{and} \quad h(x) = c_{s-1}x^{s-1} + \cdots + c_0$$

be polynomials in $\mathbb{C}[x]$ with $s \leq r$. Let $n = 2r$, and rewrite $g(x)$ and $h(x)$ with leading zeroes as

$$g(x) = b_{n-1}x^{n-1} + \cdots + b_0 \quad \text{and} \quad h(x) = c_{n-1}x^{n-1} + \cdots + c_0.$$

Let $f(x) = g(x)h(x)$, and note that $\deg f \leq n-1$. By $(*)$,

$$D\,\langle b_0, b_1, \ldots, b_{n-1}\rangle^T = \langle g(1), g(\omega), \ldots, g(\omega^{n-1})\rangle^T$$

and

$$D\,\langle c_0, c_1, \ldots, c_{n-1}\rangle^T = \langle h(1), h(\omega), \ldots, h(\omega^{n-1})\rangle^T.$$

$$g(x) = b_{r-1}x^{r-1} + \cdots + b_0 \quad \text{and} \quad h(x) = c_{s-1}x^{s-1} + \cdots + c_0$$

be polynomials in $\mathbb{C}[x]$ with $s \le r$. Let $n = 2r$, and rewrite $g(x)$ and $h(x)$ with leading zeroes as

$$g(x) = b_{n-1}x^{n-1} + \cdots + b_0 \quad \text{and} \quad h(x) = c_{n-1}x^{n-1} + \cdots + c_0.$$

Let $f(x) = g(x)h(x)$, and note that $\deg f \le n - 1$. By $(*)$,

$$D \langle b_0, b_1, \ldots, b_{n-1} \rangle^T = \langle g(1), g(\omega), \ldots, g(\omega^{n-1}) \rangle^T$$

and

$$D \langle c_0, c_1, \ldots, c_{n-1} \rangle^T = \langle h(1), h(\omega), \ldots, h(\omega^{n-1}) \rangle^T.$$

These can be computed with $O(n \log n)$ complex arithmetic operations.

$$g(x) = b_{r-1}x^{r-1} + \cdots + b_0 \quad \text{and} \quad h(x) = c_{s-1}x^{s-1} + \cdots + c_0$$

be polynomials in $\mathbb{C}[x]$ with $s \leq r$. Let $n = 2r$, and rewrite $g(x)$ and $h(x)$ with leading zeroes as

$$g(x) = b_{n-1}x^{n-1} + \cdots + b_0 \quad \text{and} \quad h(x) = c_{n-1}x^{n-1} + \cdots + c_0.$$

Let $f(x) = g(x)h(x)$, and note that $\deg f \leq n - 1$. By $(*)$,

$$D \langle b_0, b_1, \ldots, b_{n-1} \rangle^T = \langle g(1), g(\omega), \ldots, g(\omega^{n-1}) \rangle^T$$

and

$$D \langle c_0, c_1, \ldots, c_{n-1} \rangle^T = \langle h(1), h(\omega), \ldots, h(\omega^{n-1}) \rangle^T.$$

These can be computed with $O(n \log n)$ complex arithmetic operations. Computing $f(\omega^j) = g(\omega^j)h(\omega^j)$ for $0 \leq j \leq n-1$ takes n multiplications.

$$(**) \quad D^{-1}\langle f(1), f(\omega), \ldots, f(\omega^{n-1})\rangle^T = \langle a_0, a_1, \ldots, a_{n-1}\rangle^T$$

$g(x)$ and $h(x)$ with leading zeroes as

$$g(x) = b_{n-1}x^{n-1} + \cdots + b_0 \quad \text{and} \quad h(x) = c_{n-1}x^{n-1} + \cdots + c_0.$$

Let $f(x) = g(x)h(x)$, and note that $\deg f \leq n - 1$. By $(*)$,

$$D\,\langle b_0, b_1, \ldots, b_{n-1}\rangle^T = \langle g(1), g(\omega), \ldots, g(\omega^{n-1})\rangle^T$$

and

$$D\,\langle c_0, c_1, \ldots, c_{n-1}\rangle^T = \langle h(1), h(\omega), \ldots, h(\omega^{n-1})\rangle^T.$$

These can be computed with $O(n \log n)$ complex arithmetic operations. Computing $f(\omega^j) = g(\omega^j)h(\omega^j)$ for $0 \leq j \leq n-1$ takes n multiplications.

$$(**) \quad D^{-1}\langle f(1), f(\omega), \ldots, f(\omega^{n-1})\rangle^T = \langle a_0, a_1, \ldots, a_{n-1}\rangle^T$$

$g(x)$ and $h(x)$ with leading zeroes as

$$g(x) = b_{n-1}x^{n-1} + \cdots + b_0 \quad \text{and} \quad h(x) = c_{n-1}x^{n-1} + \cdots + c_0.$$

Let $f(x) = g(x)h(x)$, and note that $\deg f \leq n - 1$. By $(*)$,

$$D\,\langle b_0, b_1, \ldots, b_{n-1}\rangle^T = \langle g(1), g(\omega), \ldots, g(\omega^{n-1})\rangle^T$$

and

$$D\,\langle c_0, c_1, \ldots, c_{n-1}\rangle^T = \langle h(1), h(\omega), \ldots, h(\omega^{n-1})\rangle^T.$$

These can be computed with $O(n \log n)$ complex arithmetic operations. Computing $f(\omega^j) = g(\omega^j)h(\omega^j)$ for $0 \leq j \leq n-1$ takes n multiplications. Applying $(**)$ allows us to compute the coefficients of $f(x)$ with $O(n \log n)$ further arithmetic operations.

$$(**) \quad D^{-1}\langle f(1), f(\omega), \ldots, f(\omega^{n-1})\rangle^T = \langle a_0, a_1, \ldots, a_{n-1}\rangle^T$$

$g(x)$ and $h(x)$ with leading zeroes as

$$g(x) = b_{n-1}x^{n-1} + \cdots + b_0 \quad \text{and} \quad h(x) = c_{n-1}x^{n-1} + \cdots + c_0.$$

Let $f(x) = g(x)h(x)$, and note that $\deg f \leq n-1$. By $(*)$,

$$D\,\langle b_0, b_1, \ldots, b_{n-1}\rangle^T = \langle g(1), g(\omega), \ldots, g(\omega^{n-1})\rangle^T$$

and

$$D\,\langle c_0, c_1, \ldots, c_{n-1}\rangle^T = \langle h(1), h(\omega), \ldots, h(\omega^{n-1})\rangle^T.$$

These can be computed with $O(n \log n)$ complex arithmetic operations. Computing $f(\omega^j) = g(\omega^j)h(\omega^j)$ for $0 \leq j \leq n-1$ takes n multiplications. Applying $(**)$ allows us to compute the coefficients of $f(x)$ with $O(n \log n)$ further arithmetic operations. Rescaling, we deduce that the product of two polynomials of degree $\leq n$ in $\mathbb{C}[x]$ can be computed with $O(n \log n)$ complex additions and multiplications.

Integer Multiplication

Suppose we wish to multiply two positive integers A and B.

Integer Multiplication

Suppose we wish to multiply two positive integers A and B. Write each in binary as

$$A = (b_{r-1} \dots b_1 b_0)_2 \quad \text{and} \quad B = (c_{s-1} \dots c_1 c_0)_2.$$

Integer Multiplication

Suppose we wish to multiply two positive integers A and B. Write each in binary as

$$A = (b_{r-1} \ldots b_1 b_0)_2 \quad \text{and} \quad B = (c_{s-1} \ldots c_1 c_0)_2.$$

The idea then is to take $g(x)$ and $h(x)$ as above, and compute the product $f(x) = g(x)h(x)$.

Integer Multiplication

Suppose we wish to multiply two positive integers A and B. Write each in binary as

$$A = (b_{r-1} \dots b_1 b_0)_2 \quad \text{and} \quad B = (c_{s-1} \dots c_1 c_0)_2.$$

The idea then is to take $g(x)$ and $h(x)$ as above, and compute the product $f(x) = g(x)h(x)$. Observe that the coefficients of $f(x)$ are not in $\{0, 1\}$.

Integer Multiplication

Suppose we wish to multiply two positive integers A and B. Write each in binary as

$$A = (b_{r-1} \ldots b_1 b_0)_2 \quad \text{and} \quad B = (c_{s-1} \ldots c_1 c_0)_2.$$

The idea then is to take $g(x)$ and $h(x)$ as above, and compute the product $f(x) = g(x)h(x)$. Observe that the coefficients of $f(x)$ are not in $\{0, 1\}$. However, the product AB can still be obtained by computing $f(2)$ which, after computing the coefficients of $f(x)$, amounts to some shifts and additions that do not require much time.

Suppose we wish to multiply two positive integers A and B. Write each in binary as

$$A = (b_{r-1} \dots b_1 b_0)_2 \quad \text{and} \quad B = (c_{s-1} \dots c_1 c_0)_2.$$

The idea then is to take $g(x)$ and $h(x)$ as above, and compute the product $f(x) = g(x)h(x)$. Observe that the coefficients of $f(x)$ are not in $\{0, 1\}$. However, the product AB can still be obtained by computing $f(2)$ which, after computing the coefficients of $f(x)$, amounts to some shifts and additions that do not require much time.

This involves arithmetic operations with roots of unity.

Suppose we wish to multiply two positive integers A and B. Write each in binary as

$$A = (b_{r-1} \dots b_1 b_0)_2 \quad \text{and} \quad B = (c_{s-1} \dots c_1 c_0)_2.$$

The idea then is to take $g(x)$ and $h(x)$ as above, and compute the product $f(x) = g(x)h(x)$. Observe that the coefficients of $f(x)$ are not in $\{0, 1\}$. However, the product AB can still be obtained by computing $f(2)$ which, after computing the coefficients of $f(x)$, amounts to some shifts and additions that do not require much time.

This involves arithmetic operations with roots of unity. In 1968, Volker Straussen showed one can approximate the roots of unity to sufficient accuracy to obtain a total complexity for the multiplication of order $n \log n \, (\log \log n)^{1+\varepsilon}$ (for any fixed $\varepsilon > 0$) where n is a bound on the number of bits of A and B.

Suppose we wish to multiply two positive integers A and B. Write each in binary as

$$A = (b_{r-1} \ldots b_1 b_0)_2 \quad \text{and} \quad B = (c_{s-1} \ldots c_1 c_0)_2.$$

The idea then is to take $g(x)$ and $h(x)$ as above, and compute the product $f(x) = g(x)h(x)$. Observe that the coefficients of $f(x)$ are not in $\{0, 1\}$. However, the product AB can still be obtained by computing $f(2)$ which, after computing the coefficients of $f(x)$, amounts to some shifts and additions that do not require much time.

This involves arithmetic operations with roots of unity. In 1968, Volker Straussen showed one can approximate the roots of unity to sufficient accuracy to obtain a total complexity for the multiplication of order $n \log n \, (\log \log n)^{1+\varepsilon}$ (for any fixed $\varepsilon > 0$) where n is a bound on the number of bits of A and B. But there is an alternative approach that avoids computations of the complex roots of unity.

Main Idea: Replace complex roots of unity with roots of 1 modulo some prime (or various primes) p.

Let d be a positive integer with only small prime divisors.

Main Idea: Replace complex roots of unity with roots of 1 modulo some prime (or various primes) p.

Let d be a positive integer with only small prime divisors. Let p be a prime such that $d|(p-1)$.

Main Idea: Replace complex roots of unity with roots of 1 modulo some prime (or various primes) p.

Let d be a positive integer with only small prime divisors. Let p be a prime such that $d|(p-1)$. Then there is a positive integer ω such that ω has order d modulo $p-1$.

Main Idea: Replace complex roots of unity with roots of 1 modulo some prime (or various primes) p.

Let d be a positive integer with only small prime divisors. Let p be a prime such that $d|(p-1)$. Then there is a positive integer ω such that ω has order d modulo $p-1$. The following lemma replaces our previous one.

Lemma. *Let t be an integer. With the notation above,*

$$\sum_{j=0}^{d-1} \omega^{tj} \equiv \begin{cases} 0 \pmod{p} & \text{if } t \not\equiv 0 \pmod{d} \\ d \pmod{p} & \text{if } t \equiv 0 \pmod{d}. \end{cases}$$

Comment: Both d and ω have inverses modulo p.

Main Idea: Replace complex roots of unity with roots of 1 modulo some prime (or various primes) p.

Let d be a positive integer with only small prime divisors. Let p be a prime such that $d|(p-1)$. Then there is a positive integer ω such that ω has order d modulo $p-1$. The following lemma replaces our previous one.

Lemma. *Let t be an integer. With the notation above,*

$$\sum_{j=0}^{d-1} \omega^{tj} \equiv \begin{cases} 0 \pmod{p} & \text{if } t \not\equiv 0 \pmod{d} \\ d \pmod{p} & \text{if } t \equiv 0 \pmod{d}. \end{cases}$$

Comment: Both d and ω have inverses modulo p. One can use the discrete Fourier transform modulo p using ω as our root of unity.

Main Idea: Replace complex roots of unity with roots of 1 modulo some prime (or various primes) p.

Let d be a positive integer with only small prime divisors. Let p be a prime such that $d|(p-1)$. Then there is a positive integer ω such that ω has order d modulo $p-1$. The following lemma replaces our previous one.

Lemma. *Let t be an integer. With the notation above,*

$$\sum_{j=0}^{d-1} \omega^{tj} \equiv \begin{cases} 0 \pmod{p} & \text{if } t \not\equiv 0 \pmod{d} \\ d \pmod{p} & \text{if } t \equiv 0 \pmod{d}. \end{cases}$$

Comment: Both d and ω have inverses modulo p. One can use the discrete Fourier transform modulo p using ω as our root of unity. Both $D(d, \omega)$ and its inverse $D\big(d, \omega^{-1}\big)/d$ are defined modulo p.

Main Idea: Replace complex roots of unity with roots of 1 modulo some prime (or various primes) p.

Let d be a positive integer with only small prime divisors. Let p be a prime such that $d|(p-1)$. Then there is a positive integer ω such that ω has order d modulo $p-1$. The following lemma replaces our previous one.

Lemma. *Let t be an integer. With the notation above,*

$$\sum_{j=0}^{d-1} \omega^{tj} \equiv \begin{cases} 0 \pmod{p} & \text{if } t \not\equiv 0 \pmod{d} \\ d \pmod{p} & \text{if } t \equiv 0 \pmod{d}. \end{cases}$$

Comment: Both d and ω have inverses modulo p. One can use the discrete Fourier transform modulo p using ω as our root of unity. Both $D(d,\omega)$ and its inverse $D\left(d,\omega^{-1}\right)/d$ are defined modulo p. The previous arguments all carry through.

What's next?

We will go over a number of "Practice Problems" for the test.

Then we will go over some interesting computational related questions on irreducibility/factoring in $\mathbb{Z}[x]$.

Examples of questions we would like to answer:

1. How does

$$f(x) = 1 + x^{211} + x^{517} + x^{575} + x^{1245} + x^{1398}$$

factor in $\mathbb{Z}[x]$?

2. Let $f_0(x) = 1$. For $k \geq 1$, define $f_k(x)$ to be the reducible polynomial of the form $f_{k-1}(x) + x^n$ with n as small as possible and $n > \deg f_{k-1}$.

2. Let $f_0(x) = 1$. For $k \geq 1$, define $f_k(x)$ to be the reducible polynomial of the form $f_{k-1}(x) + x^n$ with n as small as possible and $n > \deg f_{k-1}$.

$$1$$

$$1 + x^3$$

$$1 + x^3 + x^{15}$$

$$1 + x^3 + x^{15} + x^{16}$$

$$1 + x^3 + x^{15} + x^{16} + x^{32}$$

$$1 + x^3 + x^{15} + x^{16} + x^{32} + x^{33}$$

$$1 + x^3 + x^{15} + x^{16} + x^{32} + x^{33} + x^{34}$$

$$1 + x^3 + x^{15} + x^{16} + x^{32} + x^{33} + x^{34} + x^{35}$$

Is the sequence $\{f_k(x)\}$ an infinite sequence?

3. The polynomial $f(x) = x^2 + x + 1$ is Eisenstein because one can use

$$f(x+1) = x^2 + 3x + 3$$

to deduce that $f(x)$ is irreducible by Eisenstein's criterion.

3. The polynomial $f(x) = x^2 + x + 1$ is Eisenstein because one can use

$$f(x+1) = x^2 + 3x + 3$$

to deduce that $f(x)$ is irreducible by Eisenstein's criterion. Which of the following are Eisenstein?

$$x^{15} + 2\,x^{14} - 3\,x^8 + x^7 - 3\,x + 1$$

and/or

$$x^9 - x^8 + 2\,x^5 + x^4 + 2\,x + 1$$