Number theoretic algorithms for cryptographic applications

Sandeep Sen[1]

March 16, 2009

[1]Department of Computer Science and Engineering, IIT Delhi, New Delhi 110016, India.
E-mail:ssen@cse.iitd.ernet.in

Contents

Preface

The topics covered here are supplmentary material for a course in Cryptography that I am co-teaching with Palash Sarkar. Much of the foundational basis of modern cryptography requires understanding of computational aspects of algebra and number theory and it is an attempt to cover some of the basics.

Sandeep Sen
January 2009

Chapter 1

Modular Arithmetic

In this chapter, we will discuss some useful properties of numbers when calculations are done modulo n, where $n > 0$. In the context of computer science, n is usually a power of 2 since representation is binary.

1.1 Divisibility

Definition 1.1 *An integer b is divisible by an integer a ($a \neq 0$), if there is an integer x such that $b = ax$. This will be denoted by $a|b$.*

We begin by formalising some elementary observations about integer division.

Theorem 1.1 *1. $a|b$ implies $a|bc$ for any integer c.*

2. $a|b$ and $b|c$ implies $a|c$.

3. $a|b$ and $a|c$ implies $a|bx + cy$.

4. if $m \neq 0$ then $a|b \equiv ma|mb$.

Theorem 1.2 *Given integers a and b with $a > 0$, there exist* **unique** *integers q and r such that $b = qa + r$, $0 \leq r < a$.*

The beginning of number theory goes back to Euclid's algorithm that exploited some of the properties of divisibility to compute the gcd of two integers. The gcd of two numbers a and b is the largest among the common divisors of a and b. If this is 1 then a, b are *relatively prime*. From property 3, it follows that for any common divisor c, i.e. $c|a$ and $c|b$, then $c|(b - a)$ where $b \geq a$. By repeatedly applying this, i.e., by subtracting qa, till $0 \leq b - qa < a$, we must now find the gcd of a and $b - qa$. If $a|b$, then clearly a is the gcd, so that can be used as a terminating case.

A more formal way of stating Euclid's algorithm is writing out the following equations

$$
\begin{aligned}
b &= cq_1 + r_1 && (1.1.1)\\
c &= r_1q_2 + r_2 && (1.1.2)\\
r_1 &= r_2q_3 + r_3 && (1.1.3)\\
.. &= && (1.1.4)\\
r_{j-3} &= r_{j-2}q_{j-1} + r_{j-1} && (1.1.5)\\
r_{j-2} &= r_{j-1}q_j + r_{j-1} && (1.1.6)\\
r_{j-1} &= r_jq_{j+1} + 0 && (1.1.7)\\
& && (1.1.8)
\end{aligned}
$$

Extended Euclid's algorithm states that r_j must be expressed as a linear combination of b and c, i.e. , $bx + cy = r_j$. By eliminating r_{j-2} from the last two equations, we obtain

$$r_{j-2}(1 + q_{j-1} - r_{j-3}q_{j-1} = r_j$$

Now a similar mechanism can be applied to eliminate r_{j-3} and express r_j as a linear combination of r_{j-3} and r_{j-4} until we have only b and c.

In the forward computation, numbers x and y such that gcd $= ax + by$. For this, we maintain an invariant that $ax_i + by_i = r_i$ where r_i is the remainder in the i-th iteration with initial values $x_0 = 1$ and $y_0 = 0$. The correctness of the algorithm follows from induction.

The following properties of $gcd(x, y)$ are known

Theorem 1.3 *(i) If c is a common divisor of a, b, then $c|gcd(a, b)$.*
(ii) $gcd(x, y) = \min\{ax + by\}$ where x, y are integers, such that $ax + by > 0$.
(iii) $m \cdot gcd(a, b) = gcd(ma, mb)$.
(iv) If $gcd(a, m) = gcd(b, m) = 1$ then $gcd(ab, m) = 1$.
(v) If $c|ab$ and $gcd(b, c) = 1$ then $c|a$.

Prime numbers (with no divisors other than 1 and the number itself) are extremely important in the area of number theory. Every number $n > 1$ can be expressed as a product of primes which may or may not be distinct. The fundamental theorem of arithmetic or the unique factorization states that

Theorem 1.4 *The factoring of any integer $n > 1$ is unique except for the order of the prime factors.*

Proof: We know that if $p|q_1q_2$ where p is prime then either $p|a$ or $p|b$ or both. □
The fact that number of primes is infinite was given in an elegant proof of Euclid.

Extending his argument it can be shown that there are arbitrary gaps between two primes. The *prime number theorem* says that among the first n integers there are very nearly $\frac{n}{\ln n}$ prime numbers.

1.2 Congruences

The notion of *relatively prime* is also very important, namely b and c are relatively prime if they do not have any common factors other than 1. The set of numbers relatively prime to a number n form a multiplicative *group* denoted by Z_n^*. This implies that for any numbers $x, y \in Z_n^*$, the product $x \cdot y$ modulo n is unique. Table **??** shows Z_6^*. Z_p^*, where p is prime have a nice structure and is useful in many contexts.

Definition 1.2 *If an integer m, not zero, divides the d divides the difference $a - b$, we say that a is* **congruent** *to b* **modulo** *m and is denoted by $a \equiv b(\mod m)$.*

(Since $m|(a-b)$ is equivalent to $-m|(a-b)$, we will always assume that $m > 0$.)

The following properties follow from the previous definition.

Theorem 1.5 *1. $a \equiv b(\mod m)$ is the same as $a - b \equiv 0(\mod m)$.*

2. $a \equiv b(\mod m)$ and $b \equiv c(\mod m)$ implies $a \equiv c(\mod m)$. (transitive - infact $\equiv (\mod m)$ is an equivalence relation).

3. If $a \equiv b(\mod m)$ and $c \equiv d(\mod m)$ then $ax + cy \equiv bx + dy(\mod m)$

4. If $a \equiv b(\mod m)$ and $c \equiv d(\mod m)$, then $ac \equiv bd(\mod m)$

5. If $a \equiv b(\mod m)$ and $d|m$, $d > 0$, then $a \equiv b(\mod d)$.

The degree of a polynomial (with integral coefficients) depends on the modulus over where it is computed. It is the highest power of x for which the coeffient is non-zero modulo m. For $f(x) = a_0x^n + a_1x^{n-1} + \ldots a_n$, if $f(u) \equiv 0(\mod m)$ then we say that u is a solution of the *congruence* $f(x) \equiv 0(\mod m)$. It is known that

Theorem 1.6 *If $a \equiv b(\mod m)$, then $f(a) \equiv f(b)(\mod m)$*

An important problem is the solution of congruences and in particular linear (degree 1) congruence. Any such congruence has the form

$$ax \equiv b(\mod m)$$

For the special case that $gcd(a, m) = 1$, we have a solution $x_1 = a^{\phi(m)-1}b$, where $\phi(m)$ is the *totient* function (defined by Euler). It is the number of integers less than m that are relatively prime to m (if m is prime then $\phi(m) = m - 1$). This follows from the following theorem of Euler.

Theorem 1.7 *If* $gcd(a, m) = 1$, *then* $a^{\phi(m)} \equiv 1(\mod m)$.

Another way of viewing the solution is to multiply both sides by a number a^{-1} such that $a \cdot a^{-1} \equiv 1(\mod m)$. We have the following equivalent of cancellation laws

Theorem 1.8 *1. If* $ax \equiv ay(\mod m)$ *and* $gcd(a, m) \equiv 1(\mod m)$ *then* $x \equiv y(\mod m)$.

2. $ax \equiv ay(\mod m)$ *iff* $x \equiv y(\mod \frac{m}{gcd(a,m)})$. *(generalization)*

The remaining solutions (when $gcd(a, m) = 1$) are of the form $x_1 + jm$ for any integer j. In other words there is a unique solution modulo m. For the other case (when a and m are not relatively prime), the solutions are described by the following theorem.

Theorem 1.9 *Let* $g = gcd(a, m)$. *Then* $ax \equiv b(\mod m)$ *has no solutions if* g *does not divide* b. *If* $g|b$, *it has* g *solutions* $x \equiv (b/g)x_0 + t(m/g),\ t = 0, 1 \ldots g - 1$, *where* x_0 *is any solution of* $(a/g)x \equiv (\mod (m/g)$.

Algorithmically, in both cases, we can use the (extended) Euclid's algorithm to compute x_1 or x_0. Let $ax' + my' = g$ which implies that $ax' \equiv g(\mod m)$. If $g = 1$, then clearly $x = x' \cdot b$ or $a^{-1} \equiv x'(\mod m)$. Moreover $g|b$ for the equation to be feasible. This follows from the observation that a solution exists for only those b that belong to the *cyclic subgroup* of a. One can verify easily that the cyclic subgroup $< a >=< d >= \{0, 1, 2 \ldots m/d - 1\}$

An alternate method is to solve a set of simultaneous congruences by factorising $m = \prod_{i=1}^{k} p_i^{e_i} = \prod_{i=1}^{k} m_i$ where $m_i = p_i^{e_i}$. Since m_i are relatively prime in pairs, it can be shown that the solution of the equation $ax \equiv b(\mod m)$ is the same as solving the congruences $ax \equiv b(\mod m_i)$ simultaneously for all i. Suppose the individual congruences have solutions

$$x \equiv a_i(\mod m_i)$$

Then these can be combined using a result called *Chinese Remaindering Theorem*.

Theorem 1.10 *The common solution is given by*

$$x_0 = \sum_{j=1}^{r} \frac{m}{m_j} b_j a_j$$

where b_j *is given by solutions to* $(m/m_j)b_j \equiv (\mod m_j)$.

1.2.1 The totient function

The number of elements that relatively prime to n is called Euler's totient function and denoted by $\phi(n)$. It is known that

$$\phi(n) = n \cdot \prod_{i=1}^{k}(1 - 1/p_1) \cdot (1 - 1/p_2) \ldots 1 - p_k)$$

where p_i are distinct prime factors of n. Euler's toient function has some nice properties that we state without complete proofs.

1. $\sum_{d:d|n} \phi(d) = n$
2. If p is a prime, let O_k denote the set of elements in Z_p^* that have *order* k. Then $|O_k|$ equals $\phi(k)$. In particular $|O_{p-1}| = \phi(k-1)$.
3. $\phi(n)$ is $\Omega(n/\log n)$ that follows from the definition of the totient function and minimising it.

Notice that the third one implies a Monte Carlo randomized algorithm for finding a *generator* of Z_n^*.

1.2.2 Quadratic residues and non-trivial square roots

If $x^2 \equiv a \mod n$ then a is a *quadratic residue* and x is a *square root* of a. When n is prime there are exactly two square roots.

Theorem 1.11 (Euler's criterion) *If n is prime, then $a^{n-1/2} \equiv 1$ iff a is a quadratic residue.*
Moreover, there are exactly $(n-1)/2$ quadratic residues.

proof Suppose $x^2 \equiv a \mod n$. Then $(x^2)^{(n-1)/2} = x^{n-1} \equiv 1 \mod n$.
For the other direction,let g be a generator (since n is prime there exists one) and let $a = g^k$. So $(g^k)^{(n-1)/2} \equiv 1 \mod n$. Then $k \cdot (n-1)/2$ must be multiple of $n-1$, implying k is even and therefore $g^{k/2}$ is a square-root of a.

The following result is stated without proof.

Theorem 1.12 *In Z_p^* where p is a prime, the square root of a quadratic residue can be found in randomized polynomial time, given a known* **non-residue**.

A non-trivial square roots is an element $y \neq \{-1, 1\}$ such that $y^2 \equiv 1$. The following observation is exploited in primality testiing.
Observation: If nontrivial square root exists in Z_n^* then n is prime.

Since $x^2 \equiv 1 \mod n$, $(x+1)(x-1) \equiv 0 \mod n$. Given that $x \neq \{1, -1\}$, the l.h.s. must be a multiple of n and therefore both $gcd(x+1, n)$ and $gcd(x-1, n)$ must be greater than 1 and hence we have a factor of n.

Chapter 2

Rabin-Miller test

Fermat's theorem is a 1-sided test, namely, if $a^{n-1} \not\equiv 1 \mod n$, then n is composite. For the converse, it is not guaranteed to work and in some cases called **Carmichael numbers**, for all $1 < a < n$, $a^n \equiv 1 \mod n$. Fortunately, there are very few (even though infinite) such numbers in terms of density (fraction of all integers). But, if we don't want the test to be input-dependent, we cannot afford to err on Carmichael numbers.

Here is some good news.

Observation If there exists at least one $1 < a < n$ such that $a^{n-1} \not\equiv 1 \mod n$, then less than half the numbers will satisfy $x^{n-1} \equiv 1 \mod n$.

Proof Consider an a such that $a^{n-1} \not\equiv 1 \mod n$, then for any b $b^{n-1} \equiv 1 \mod n$, we obtain $c = a \cdot b$, such that $c^{n-1} \not\equiv 1 \mod n$. So the fraction of elements of Z_n^* that do not pass Fermat test is at least half. This can also be argued by observing that $\{x | x^{n-1} \equiv 1 \mod n\}$ is a proper subgroup of Z_n^*.

Also note that for any element $y \notin Z_n^*$, $y^{n-1} \not\equiv 1 \mod n$.

Rabin Miller primality testing
Let $n - 1 = 2^k \cdot m$ where m is odd.

1. Choose an element a uniformly at random from $[2, n-1]$.
2. If $b = a^m \equiv 1 \mod n$ then **n is prime**
3. for i:= 0 to $k - 1$ do

   If $b \equiv -1 \mod n$ then **n is prime**
   $b := b * b \mod n$
4. Return **n is composite**

The intuition behind this approach is that we may find a non-trivial root of 1 which can be used to find a common factor if n is composite.

The running time of this algorithm is clearly polynomial - in fact $O(\log^3 n)$ steps for a $\log n$ bit integer.

Broadly, we will prove the following

Theorem 2.1 *The primality testing algorithm of Rabin-Miller will satisfy the following property*
(i) If the input is prime then it will output the right answer.
(ii) If the input is composite, it will answer correctly with probability at least 1/2 (irrespective of the input).

This kind of algorithm is called Monte Carlo randomized algorithm and the probability of error for a composite input can be reduced to $1/2^k$ by running the test independently k times.

Proof Let us denote the sequence of values assigned to b as the k tuple

$$B(a) = (b_0 = a^m, a^{2m}, a^{2^2 m}, \ldots b_{k-1} = a^{m \cdot 2^{k-1}})$$

Clearly once $b_j \equiv \{-1, 1\}$, subsequently it is 1. Moreover, if the sequence ends with 1 without ever becoming -1, then we have discovered a non-trivial square root of 1 implying n is composite. The proof is by case analysis on the following

Case 1 n is prime: Since there is no non-trivial square root of 1, the tuple must have an occurence of -1 since $a^{n-1} \equiv 1$. Alternately, $a^m \equiv 1$, in which case it doesn't enter the loop and declares n to be prime in the first test. So, it always returns the correct answer.

Case 2 n is a non-Carmichael composite: From the observation preceding the algorithm, for at least half of the choices of a, $a^{n-1} \not\equiv 1$. So, with probability at least $1/2$, the tuple will not contain -1 otherwise in the next step, we will get 1 and therefore $a^{n-1} \equiv 1$. Note that we are not even checking if $a^{n-1} \equiv 1$ since in either case n is composite.

Case 3 n is a Carmichael composite: This is the crux of the algorithm, since for all a, $a^{n-1} \equiv 1$. Therefore we have to analyse the tuple B more closely for Carmichael numbers. So we want to find out the number of choices (witnesses) for which the tuple **does not** contain -1, i.e., we obtain a non-trivial square root.

Claim If $x^{n-1} \equiv 1 \mod n$ then n cannot be a power of odd prime, i.e. $n \neq p^e$.

Proof is ommitted from here and instead we make another claim which makes the previous result somewhat redundant in the context of primality.

Claim There is a polynomial time algorithm to find out if $n = x^y$ for some integers $x, y \geq 2$.

Proof is left as an exercise. So from this point we can assume that $n = n_1 \cdot n_2$ where $n_1 \neq n_2$ (may not be unique) and relatively prime to eachother.

Consider the elements $W = x \in Z_n^* \ : \ x^{2^j m} \equiv \pm 1$ and for which $j = j'$ is as large as possible (note that $(-1)^{2^0 m} \equiv -1$).

W is closed under multiplication and therefore it is a subgroup[1] and therefore if we can show that W is a proper subgroup of Z_n^*i, at least half of the elements $y \notin W$ are such that $y^{2^{j'} m} \not\equiv \pm 1$. Either $y^{n-1} \equiv 1$ implying that we have discovered a non-trivial square root or $y^{n-1} \not\equiv 1$ again implying that n is composite.

Proof that W is a proper subgroup

Lect $w \equiv -1 \mod n_1$ and $w \equiv 1 \mod n_2$. Then $w \not\equiv \pm 1 \mod n$ (prove by contradiction - note that if $w \equiv 1 \mod n$ then $w \equiv 1 \mod n_1$ and $w \equiv \mod n_2$). Moreover, $w \in Z_n^*$ since w doesn't have common factor with both n_1 and n_2.

2.1 Legendre symbol and computing Jacobi

The **Legendre symbol** of $a \in Z_p^*$ is defined as

$$\left[\frac{a}{p}\right] \equiv a^{(p-1)/2} \mod p$$

which is +1 if a is a quadratic residue and -1 otherwise (Euler criterion).

The generalization to arbitrary **odd** n is called **Jacobi symbol** and is defined for $a \in Z_n^*$ (i.e. a is relatively prime to n) as

$$\left[\frac{a}{n}\right] = \prod_{i=1}^{t} \left[\frac{a}{p_i}\right]^{k_i}$$

where $n = p_1^{k_1} \cdot p_2^{k_2} \ldots p_t^{k_t}$ is the prime factorization of n.

The Solovay-Strassen primality testing algorithm involves computation of Jacobi *without* prime factorization of n. The following properties are known about Jacobi symbol.

1. $\left[\frac{ab}{n}\right] = \left[\frac{a}{n}\right] \cdot \left[\frac{b}{n}\right]$
2. For $a \equiv b \mod n$, $\left[\frac{a}{n}\right] = \left[\frac{b}{n}\right]$
3. For odd co-prime a, n

$$\left[\frac{a}{n}\right] = (-1)^{(a-1)/2 \cdot (n-1)/2} \cdot \left[\frac{n}{a}\right]$$

[1] Any closed subset of a group is a subgroup

4. $\left[\frac{1}{n}\right] = 1$

5. $\left[\frac{a}{n}\right] = -1$ for $n \equiv \{3, 5\} \mod 8$.
   and is equal to 1 for $n \equiv \{1, 7\} \mod 8$.

The last two properties can be thought of as base cases.

Exercise Design an efficient (polynomial time) algorithm for computing the Jacobi symbol of a given number.

Chapter 3

Polynomial factorization

Polynomials can be thought of as an ordered sequence of coefficients a_i $0 \leq i \leq n$ where k is the **degree** of a polynomial if $a_k \neq 0$ and $a_{k+1}, a_{k+2} \ldots a_n = 0$. The coefficient a_i is associated with x^i which is an indeterminate or can be thought of as a *placeholder*. The coefficients $a_i \in Z_p^*$ where p is a prime. We will actually use the properties of the field $F_p = \{0, 1 \ldots p-1\}$.

3.1 Quadratic polynomial

Wlog, we can assume that the roots are not equal. Otherwise it is an easy case and can be solved directly. Let $P(x) = x^2 + bx + c = 0$ where $b \neq 0$. Let α, β $\alpha \neq \beta$ be the two roots (if they exist). Suppose α, β are such that one is a quadratic indent**Exercise** Provide details of Step 4 and analyse. residue[1] (modulo p) and the other is not. Since every non-quadratic residue q satisfies $q^{p-1/2} \equiv 1$, it is a root of the polynomial $X^{p-1/2} - 1$. Therefore the gcd of $(P(x), X^{p-1/2} - 1)$ yields a non-trivial factor.
Note : If both roots are q.r., then the gcd is the polynomial itself.

To address the general case, we will try to do a *random shift* of the roots, say by r. Let $P_r(x) = (x - \alpha - r)(x - \beta - r)$, clearly if we can find the roots of $P_r(x)$, we can find the roots of $P(x)$. Also note that $P_r(x) = P_r(x) - 2rx + r^2 + br$, i.e., the coefficients can be easily computed. Let us now analyse the Legendre symbols of the roots $\alpha + r$ and $\beta + r$ for a randomly chosen r.

Lemma 3.1 *For any $\alpha, \beta \in Z_p^*$ $\alpha \neq \beta$, the probability that $\left[\frac{\alpha+r}{p}\right] = \left[\frac{\beta+r}{n}\right]$ is less than 1/2 for a randomly chosen r in $Z_p^* - \{\alpha, \beta\}$.*

Proof Let $s \in Z_p^*$, $s \neq 0$ be such that

$$(\alpha + r) \cdot s \equiv_p a \text{ and } (\beta + r) \cdot s \equiv_p b$$

[1] corresponding to Legendre symbol being +1 or -1

for some $a, b \in Z_p^*$. By Substituting $t = r \cdot s$, we can solve the simultaneous equations over the field in s and t, where $s = (a-b) \cdot (\alpha - \beta)^{-1}$ and $t = a - \alpha \cdot s$. Note that this gives a solution for r also provided $s \neq 0$ ($r = t \cdot s^{-1}$). If $a \neq b$, $s \neq 0$, and for every pair (a, b), there is a soln (s, r). So there is an bijection between (r, s) pairs and (a, b) pairs and uniform distribution on (r, s) implies uniform distribution in the (a, b) pairs, *conditioned on* $s \neq 0$. It can be easily seen (from symmetry) that the probability that $\left[\frac{a}{n}\right] \neq \left[\frac{b}{n}\right]$ is greater than $1/2$. Therefore, given $s \neq 0$,

$$\Pr\left\{\left[\frac{(\alpha+r)s}{n}\right] \neq \left[\frac{(\beta+r)s}{n}\right]\right\} = \Pr\left\{\left[\frac{a}{n}\right] \neq \left[\frac{b}{n}\right]\right\} > \frac{1}{2}$$

Since $\left[\frac{sy}{n}\right] = \left[\frac{s}{n}\right] \cdot \left[\frac{y}{n}\right]$ for $y \in Z_p^*$,

$$\Pr\left\{\left[\frac{(\alpha+r)s}{n}\right] \neq \left[\frac{(\beta+r)s}{n}\right]\right\} = \Pr\left\{\left[\frac{(\alpha+r)}{n}\right] \neq \left[\frac{(\beta+r)}{n}\right]\right\} > \frac{1}{2}$$

For events A, B, $\Pr[A] = \Pr[A \cap \bar{B}] + \Pr[A \cap B] \geq \Pr[A \cap B] = \Pr[A|B] \cdot \Pr[B]$ There, we can uncondition the above calculation by defining event B to be $s \neq 0$ and the event A as $\left[\frac{(\alpha+r)}{n}\right] \neq \left[\frac{(\beta+r)}{n}\right]$. Using $\Pr[s \neq 0] = 1 - 1/p$, we arrive at the required result. The complete algorithm for factorization of a quadratic polynomial is given below.

Quadpolyroot

Input: non-irreducible, monic , square-free degree 2 polynomial $P(x) : x^2 + \alpha x + \beta \mod p$

(Comment: the other cases can be handled directly)

Output: the distinct roots r_1, r_2 of the input polynomial

1. choose an r uniformly at random from $\{0, 1, \ldots (p-1)\}$
2. compute the shifted polynomial $P_r(x) = x^2 + \alpha' x + \beta'$.
3. if $\beta' = 0$ then $r_1 = -r,\ r_2 = -r - \alpha'$
4. Compute $u(x) = gcd(P_r(x), x^{(p-1)/2} - 1)$ using repeated squaring inside the Euclid's algorithm.
5. If $u(x) = P_r(x)$ or 1 (failed to find linear factors), go to step 1.
6. If $u(x) = x - c$ then return $r_1 = c - r,\ r_2 = -(\alpha' + c + r)$.

Exercise 3.1 *Provide details of Step 4 and analyse.*

Exercise 3.2 *How would you handle the case of irreducible*

polynomial ?

3.2 Higher degree polynomials

The previous algorithm works specifically for degree 2 polynomials and exploits the fact. Before we take up the more general case, let us discuss some facts about polynomials whose coeffients are from F_p.

Observation 3.1 *If $v(x)$ is a polynomial with coeffients modulo a prime p,*

$$(v(x))^p = v(x^p)$$

First observe that

$$\begin{aligned}(v_1(x) + v_2(x))^p &= (v_1(x))^p + \binom{p}{1}(v_1(x))^{p-1}v_2(x) + \ldots (v_2(x))^p \\ &= (v_1(x))^p + (v_2(x))^p \text{ all other terms are multiple of } p\end{aligned}$$

Let $v(x) = v_m x^m + v_{m-1}x^{m-1} + \ldots v_0$. Then

$$\begin{aligned}(v(x))^p &= (v_m x^m + v_{m-1}x^{m-1} + \ldots v_0)^p \\ &= (v_m x^m)^p + (v_{m-1}x^{m-1})^p + \ldots (v_0 x^0)^p \\ &= v_m^p x^{mp} + v_{m-1}^p x^{(m-1)p} + \ldots v_0^m \\ &= v_m x^{mp} + v_{m-1}x^{(m-1)p} + \ldots v_0 \text{ (Fermat result)} \\ &= v(x^p)\end{aligned}$$

Like degree two polynomial, let us get rid of multiple roots, in fact, repeated factors. Let $u(x) = u)nx^n + u_{n-1}x^{n-1} + \ldots u_0$. Suppose $u(x) = (v(x))^2 w(x)$. Then, by taking derivatives,

$$\begin{aligned}u'(x) &= 2v(x)v'(x)w(x) + (v(x))^2 w'(x) \\ &= = nu_n x^{n-1} + (n-1)u_{n-2}x^{n-2} + \ldots u_1\end{aligned}$$

Note that both side of the equation are multiples of $v(x)$. If $d(x) = gcd(u(x), u'(x))$ then we can claim the following

- $d(x) = 1$ Then $u(x)$ is square-free.
- $d(x) \neq 1$ then we have found a factor and we can repeat the above step to make it square-free.

If $d(x) = u(x)$, then $u'(x) = 0$,so $u(x)$ is a polynomial that has non-zero coefficients only when the corresponding exponent is a multiple of p. In other words $u(x)$ can be written as $t(x^p)$ which equals $(t(x))^p$. We can now proceed to apply the aove method to $t(x)$ and factorize $t(x)$.

3.3 Factorising a square free polynomial

Let $u(x) = p_1(x) \cdot p_2(x) \ldots p_r(x)$ where p_i 's are distinct irreducible polynomials (and therefore they do not have common factors pairwise). From Chinese Remainder Theorem [2]. there exists a unique $v(x) \mod u(x)$ such that

$$\begin{aligned} v(x) &\equiv s_1 \mod p_1(x) \\ v(x) &\equiv s_2 \mod p_2(x) \\ v(x) &\equiv s_r \mod p_r(x) \end{aligned}$$

where $s_i \in \{0, 1, p-1\}$. Note that

$$v(x)^p \equiv s_i^p \equiv_p s_i \equiv v(x) \mod p_i(x)$$

Since this holds for all i, it follows from CRT that $v(x)^p \equiv v(x) \mod u(x)$. Recall that $v(x)^p \equiv_p v(x^p)$.

Observation 3.2 *$p_i(x)$ divides $v(x)^{(p-1)/2} - 1$ iff $s_i^{(p-1)/2} \equiv_p 1$.*

From $v(x) \equiv s_i \mod p_i(x)$, it follows that $v(x)^{(p-1)/2} - 1 \equiv s_i^{(p-1)/2} - 1 \mod p_i(x)$. Since $p_i(x)$ has degree at least 1, it can only happen if $s_i^{(p-1)/2} - 1 \equiv_p 0$.

It follows that if s_i is randomly chosen, with probability 1/2, $p_i(x)$ divides $v(x)^{(p-1)/2} - 1$. So the gcd of $u(x)$ and $v(x)$ will be a multiple of $p_i(x)$ with probability 1/2.

This is the basis of the factorization algorithm. The problem is that we do not know $v(x)$ and it is not clear how to compute it since it is related to $p_i(x)$ which is what we have set out to compute.

We try to use the identity $v(x)^p \equiv_p v(x^p)$ and consequently express $x^{p \cdot j}$ in terms of x^j's. Let

$$x^{pi} \equiv q_{j,n-1}x^{n-1} + q_{j,n-2}x^{n-2} + \ldots q_{j,0} \mod u(x)$$

[2]there exists a analogous version for polynomials

$$\text{If } Q = \begin{bmatrix} q_{0,0} & q_{0,1} & \cdots & q_{0,n-1} \\ q_{1,0} & q_{1,1} & \cdots & q_{1,n-1} \\ \vdots & & & \\ q_{n-1,0} & q_{n-1,0} & \cdots & q_{n-1,0} \end{bmatrix}$$

then it can be verified that the following matrix equation holds

$$(v_0 v_1 \dots v_{n-1})Q = (v_0 v_1 \dots v_{n-1})$$

This can be seen as follows

$$v(x) = \sum_i v_i x^i = \sum_i \sum_j v_j q_{j,i} x^j = \sum_j v_j \sum_i q_{j,i} x^j = \sum_j v_j x^{pj} = v(x^p) \equiv v(x)^p \mod u(x)$$

This can be solved as

$$v(Q - I) = 0$$

using something like Gauss-Jordan iterations which also yields the nullspace in terms of an orthogonal basis $v^1, v^2 \dots v^t$. Therefore all the solutions for $v(x)$ can be expressed as

$$a_1 \cdot v^1 + a_2 \cdot v^2 + \dots a_t \cdot v^t$$

where $a_i \in [1, 2, \dots p-1]$. Note that there are p^t solutions and a **random** solution of $v(x)$ can be obtained by choosing a_is randomly and then compute $gcd(u(x), v(x)^{(p-1)/2} - 1)$.

At this point we must relate the solutions obtained from nullspace vectors to the original problem of the r irreducible factors. Clearly $t \geq r$, but we need to prove $t = r$ to make our technique efficient. If $t = r$, then we can generate a random solution of the original equations by choosing a_i's randomly. For this we need to show that every solution to $v(x)^p \equiv v(x) \mod u(x)$ must satisfy the original equations $v(x) \equiv s_i \mod p_i(x)$.

It can be shown (proof left as exercise) that

$$v(x)^p - v(x) = (v(x) - 0)(v(x) - 1) \dots (v(x) - (p-1))$$

where the terms on the RHS are relatively prime. So if $u(x)$ divides the LHS, then the irreducible factors must divide some term on the RHS, i.e., $v(x) \equiv j \mod p_i(x)$ for some $j \in [0, 1, \dots (p-1)]$ and some irredicible factor $p_i(x)$.

Chapter 4

Fast Fourier Transform and Applications

4.1 Polynomial evaluation and interpolation

A polynomial $\mathcal{P}(x)$ of degree $n-1$ in indeterminate x is a power series with maximum degree $n-1$ and has the general form $a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots a_1 x + a_0$, where a_i are coefficients over some field, typically the complex numbers $\mathbb{C}$. Some of the most common problems involving polynomials are

evaluation Given a value for the indeterminate x, say x', we want to compute $\sum_{i=0}^{n-1} a_i \cdot {x'}^i$.
By Horner's rule, the most efficient way to evaluate a polynomial is given by the formula

$$(((a_{n-1}x' + a_{n-2})x' + a_{n-3})x' + \ldots a_0$$

We are interested in the more general problem of evaluating a polynomial at multiple (distinct) points, say $x_0, x_1 \ldots x_{n-1}$. If we apply Horner's rule then it will take $\Omega(n^2)$ operations, but we will be able to do it much faster.

interpolation Given n values (not necessarily distinct), say $y_0, y_1 \ldots y_{n-1}$, there is a unique polynomial of degree $n-1$ such that $\mathcal{P}(x_i) = y_i$ x_i are distinct.
This follows from the fundamental theorem of algebra which states that a polynomial of degree d has at most d roots. Note that a polynomial is characterized by its coefficients a_i $0 \leq i \leq n-1$. A popular method for interpolation is the *Lagrange's formula.*

$$\mathcal{P}(x) = \sum_{k=0}^{n-1} y_k \cdot \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq k}(x_k - x_j)}$$

Exercise 4.1 *Show that Lagrange's formula can be used to compute the coefficients a_i's in $O(n^2)$ operations.*

One of the consequences of the interpolation is an alternate representation of polynomials as $\{(x_0, y_0), (x_1, y_1) \ldots (x_{n-1}, y_{n-1})\}$ from where the coefficients can be computed. We will call this representation as the *point-value* representation.

multiplication The product of two polynomials can be easily computed in $O(n^2)$ steps by clubbing the coefficients of the powers of x. This is assuming that the polynomials are described by their coefficients. If the polynomials are given by their point-value, then the problem is considerably simpler since

$$P(x) = P_1(x) \cdot P_2(x) \text{ where } P\text{is the product of } P_1 \text{ and } P_2$$

A closely related problem is that of *convolution* where we have to perform computations of the kind $c_i = \sum_{l+p=i} a_l \cdot b_p$ for $1 \leq i \leq n$.

The efficiency of many polynomial related problems depends on how quickly we can perform transformations between the two representations.

4.2 Cooley-Tukey algorithm

We will solve a restricted version of the evaluation problem where we will carefully choose the points $x_0, x_1 \ldots x_{n-1}$ to reduce the total number of computations, Let n be a power of 2 and let us choose $x_{n/2} = -x_0, x_{n/2+1} = -x_1, \ldots x_{n-1} = -x_{n/2-1}$. You can verify that $\mathcal{P}(x) = \mathcal{P}_E(x^2) + xP_o(x^2)$ where

$$\mathcal{P}_E = a_0 + a_2 x + \ldots a_{n-2} x^{n/2-1}$$

$$\mathcal{P}_0 = a_1 + a_3 x + \ldots a_{n-1} x^{n/2-1}$$

corresponding to the even and odd coefficients and $\mathcal{P}_E, \mathcal{P}_O$ are polynomials of degree $n/2 - 1$.

$$\mathcal{P}(x_{n/2}) = \mathcal{P}_E(x_{n/2}^2) + x_{n/2}\mathcal{P}_O(x_{n/2}^2) = \mathcal{P}_E(x_0^2) - x_0\mathcal{P}_O(x_0^2)$$

since $x_{n/2} = -x_0$. More generally

$$\mathcal{P}(x_{n/2+i}) = \mathcal{P}_E(x_{n/2+i}^2) + x_{n/2+i}\mathcal{P}_O(x_{n/2+i}^2) = \mathcal{P}_E(x_i^2) - x_i\mathcal{P}_O(x_i^2), \quad 0 \leq 0 \leq n/2 - 1$$

since $x_{n/2+i} = -x_i$. Therefore we have reduced the problem of evaluating a degree $n-1$ polynomial in n points to that of evaluating two degree $n/2-1$ polynomials at $n/2$ points $x_0^2, x_1^2 \ldots x_{n/2-1}^2$. This will also involve $O(n)$ multiplications and additions to compute the values at the original points. To continue this reduction, we have to choose points such that $x_0^2 = -x_{n/4}^2$ or equivalently $x_{n/2} = \sqrt{-1} = \omega$. This involves complex numbers if we started with coefficients in $\mathbb{R}$[1]. If we continue with

[1]Depending on our choice of the field F, we can define ω such that $\omega^2 = -1$.

this strategy of choosing points, at the j-th level of recursion, we require

$$x_i^{2^{j-1}} = -x_{\frac{n}{2^j}+i}^{2^{j-1}} \quad 0 \leq i \leq \frac{n}{2^j} - 1$$

This yields $x_1^{2^{\log n-1}} = -x_0^{2^{\log n-1}}$, i.e., if we choose $\omega^{n/2} = -1$ then $x_i = \omega x_{i-1}$. By setting $x_0 = 1$, the points of evaluation work out to be $1, \omega, \omega^2 \ldots \omega^{n/2} \ldots \omega^{n-1}$ which are usually referred to as the principal n-th roots of unity.

Analysis

Let $\mathcal{P}(x)_{a_0,a_1\ldots a_{n-1}}^{z_1,z_2\ldots z_i}$ denote the evaluation of $\mathcal{P}(x)$ with coefficients $a_0, a_1 \ldots a_{n-1}$ at points $z_1, z_2 \ldots z_i$. Then we can write the recurrence

$$\mathcal{P}(x)_{a_0,a_1\ldots a_{n-1}}^{1,\omega,\omega^2\ldots\omega^{n-1}} = \mathcal{P}(x)_{a_0,a_2\ldots a_{n/2-2}}^{1,w\ldots\omega^{n/2-1}} + \mathcal{P}(x)_{a_1,a_3\ldots a_{n/2-1}}^{1,w\ldots\omega^{n/2-1}} + O(n) \text{ multiplications and additions}$$

This immediately yields $O(n \log n)$ operations for the FFT computation.

For the inverse problem, i.e., interpolation of polynomials given the values at $1, \omega, \omega^2 \ldots \omega^{n-1}$, let us view the process of evaluation as a matrix vector product.

$$\begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega^2 & \omega^4 & \ldots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \ldots & \omega^{3(n-1)} \\ \vdots & & & & \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \ldots & \omega^{(n-1)(n-1)} \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

Let us denote this by the matrix equation $A \cdot \overline{a} = \overline{y}$. In this setting, the interpolation problem can be viewed as computing the $\overline{a} = A^{-1} \cdot \overline{y}$. Even if we had A^{-1} available, we still have to compute the product which could take $\Omega(n^2)$ steps. However the good news is that the inverse of A^{-1} is

$$\frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \frac{1}{\omega^2} & \frac{1}{\omega^4} & \ldots & \frac{1}{\omega^{2(n-1)}} \\ 1 & \frac{1}{\omega^3} & \frac{1}{\omega^6} & \ldots & \frac{1}{\omega^{3(n-1)}} \\ \vdots & & & & \\ 1 & \frac{1}{\omega^{n-1}} & \frac{1}{\omega^{2(n-1)}} & \ldots & \frac{1}{\omega^{(n-1)(n-1)}} \end{bmatrix}$$

which can be verified by multiplication with A. Recall that

$$1 + \omega^i + \omega^{2i} + \omega^{3i} + \ldots w^{i(n-1)} = 0$$

(Use the identity $\sum_j \omega^{ji} = \frac{\omega^{in}-1}{\omega^i-1} = 0$ for $\omega^i \neq 1$.)
Moreover $\omega^{-1}, \omega^{-2}, \ldots w^{-(n-1)}$ also satisfy the properties of n-th roots of unity. This enables us to use the same algorithm as FFT itself that runs in $O(n \log n)$ operations.

4.3 The butterfly network

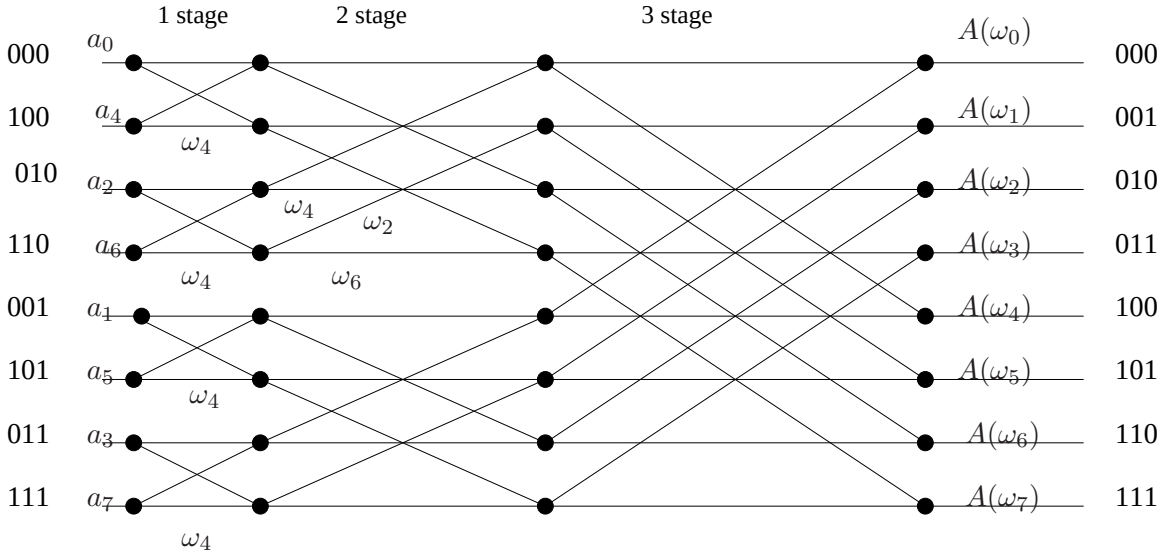

Figure 4.1: Computing an eight point FFT using a butterfly network

If you unroll the recursion of an 8 point FFT, then it looks like the Figure 4.1. Let us work through some successive recursive calls.

$$\mathcal{P}_{0,1,..7}(\omega_0) = \mathcal{P}_{0,2,4,6}(\omega_0^2) + \omega_0 \mathcal{P}_{1,3,5,7}(\omega_0^2)$$

$$\mathcal{P}_{0,1,..7}(\omega_4) = \mathcal{P}_{0,2,4,6}(\omega_0^2) - \omega_0 \mathcal{P}_{1,3,5,7}(\omega_0^2)$$

Subsequently, $\mathcal{P}_{0,2,4,6}(\omega_0^2) = \mathcal{P}_{0,4}(\omega_0^4) + w_0^2 \mathcal{P}_{2,6}(\omega_0^4)$ and

$$\mathcal{P}_{0,2,4,6}(\omega_2^2) = \mathcal{P}_{0,4}(\omega_0^4) - w_0^2 \mathcal{P}_{2,6}(\omega_0^4)$$

To calculate $\mathcal{P}_{0,4}(\omega_0^4)$ and $\mathcal{P}_{0,4}(\omega_1^4)$ we compute $\mathcal{P}_{0,4}(\omega_0^4) = \mathcal{P}_0(\omega_0^8) + \omega_0^4 \mathcal{P}_4(\omega_0^8)$ and

$$\mathcal{P}_{0,4}(\omega_1^4) = \mathcal{P}_0(\omega_0^8) - \omega_0^4 \mathcal{P}_4(\omega_0^8)$$

Since $\mathcal{P}_i$ denotes a_i, we do not recurse any further. Notice that in the above figure a_0 and a_4 are the multipliers on the left-hand side. Note that the indices of the a_i on the input side correspond to the mirror image of the binary representation of i. A *butterfly* operation corresponds to the gadget $\bowtie$ that corresponds to a pair of recursive calls. The black circles correspond to "+" and "-" operations and the appropriate multipliers are indicated on the edges (to avoid cluttering only a couple of them are indicated).

One advantage of using a network is that, the computation in each stage can be carried out in parallel, leading to a total of $\log n$ parallel stages. Thus FFT is inherently parallel and the butterfly network manages to capture the parallelism ina natural manner.

4.4 Schonage and Strassen's fast multiplication

In our analysis of the FFT algorithm, we obtained a time bound with respect to multiplication and additions in the appropriate field - implicitly we assumed $\mathbb{C}$, the complex field. This is not consistent with the boolean model of computation and we should be more careful in specifying the precision used in our computation. This is a topic in itself and somewhat out of the scope of the discussion here. In reality, the FFT computations are done using limited precision and operations like rounding that inherently result in numerical errors.

In other kinds of applications, like integer multiplication, we choose an appropriate field where we can do exact arithmetic. However, we must ensure that the field contains n-th roots of unity. Modular arithmetic, where computations are done modulo a prime number is consistent with the arithmetic done in hardware.

Observation 4.1 *In $\mathbb{Z}_m$ where $m = 2^{tn/2} + 1$ and n is a power of 2, we can use $\omega = 2^t$.*

Since n and m are relatively prime, n has a unique inverse in $\mathbb{Z}_m$ (recall extended Euclid's algorithm). Also

$$\omega^n = \omega^{n/2} \cdot \omega^{n/2} = (2^t)^{n/2} \cdot (2^t)^{n/2} \equiv (m-1) \cdot (m-1) \bmod m \equiv (-1) \cdot (-1) \bmod m \equiv 1 \bmod m$$

Claim 4.1 *If the maximum size of a coefficient is b bits, the FFT and its inverse can be computed in time proportional to $O(bn \log n)$.*

Note that addition of two b bit numbers take $O(b)$ steps and the multiplications with powers of ω are multiplications by powers of two which can also be done in $O(b)$ steps. The basic idea of the algorithm is to extend the idea of polynomial multiplication. Recall, that in Chapter **??** , we had divided each number into two parts and subsequently recursively computed by computing product of smaller numbers. By extending this strategy, we divide the numbers a and b into k parts $a_{k-1}, a_{k-2}, \ldots a_0$ and $b_{k-1}, b_{k-2}, \ldots b_0$.

$$a \times b = \left(a_{k-1} \cdot x^{k-1} + a_{k-2} \cdot x^{k-2} + \ldots a_0\right) \times \left(b_{k-1} \cdot x^{k-1} + b_{k-2} \cdot x^{k-2} + \ldots b_0\right)$$

where $x = 2^{n/k}$ - for simplicity assume n is divisible by k. By multiplying the RHS, and clubbing the coefficients of x^i, we obtain

$$a \times b = a_{k-1}b_{k-1}x^{2(k-1)} + (a_{k-2}b_1 + b_{k-2}a_1)x^{2k-3} + \ldots a_0b_0$$

Although in the final product, $x = 2^{n/k}$, we can compute the coefficients using *any* method and perform the necessary multiplcations by an appropriate power of two (which is just adding trailing 0's). This is polynomial multiplication and each term is a convolution, so we can invoke FFT-based methods to compute the coefficients. The following recurrence captures the running time

$$T(n) \leq P(k, n/k) + O(n)$$

where $P(k, n/k)$ is the time for polynomial multiplication of two degree $k-1$ polynomials involving coefficients of size n/k. (In a model where the coefficients are not too large, we could have used $O(k \log k)$ as the complexity of polynomial multiplication.) We will have to do *exact* computations for the FFT and for that we can use modular arithmetic. The modulo value must be chosen carefully so that
(i) It must be larger than the maximum value of the numbers involved, so that there is no loss of information
(ii) Should not be too large, otherwise, operations will be expensive.

Moreover, the polynomial multiplication itself consists of three distinct phases

(i) Forward FFT transform. This takes $O(bk \log k)$ using b bits.
(ii) Paiwise product of the values of the polynomials at the roots of unity. This will be done recursively with cost $2k \cdot T(b)$ where $b \geq n/k$.
The factor two accounts for the number of coefficients of the product of two polynomials of degree $k-1$.
(iii) Reverse FFT, to extract the actual coefficients. This step also takes $O(bk \log k)$ where b is the number of bits in each operand.

So the previous recurrence can be expanded to

$$T(n) \leq r \cdot T(b) + O(bk \log k)$$

where $r \cdot b \geq n$ and we must choose an appropriate value of b. For coefficients of size s, we can argue that the maximum size of numbers during the FFT computation is $2s + \log r$ bits (sum of r numbers of pairwise multiplication of s bit numbers). If we choose r to be roughly $\sqrt{n/\log n}$, then $b = \sqrt{n \log n}$ and we can rewrite the recurrence as

$$T(n) \leq 2\sqrt{\frac{n}{\log n}} \cdot T(2\sqrt{n \log n} + \log n) + O(n \log n) \tag{4.4.1}$$

Exercise 4.2 *With appropriate teminating condition, say the $O(n^{\log_2 3})$ time multiplication algorithm, verify that $T(n) \in O(n \log^2 n \log\log n)$.*

An underlying assumption in writing the recurrence is that all the expressions are integral. This can actually be ensured by choosing $n = 2^\ell$ and carefully choosing $\sqrt{n}$ for even and odd values of ℓ. Using the technique of *wrapped convolution*, one can save a factor of two in the degree of the polynomial, yielding the best known $O(n \log n \log\log n)$ algorithm for multiplication.