

17: COMPLEXITY

17.1 Polynomial Time

How long will it take to compute the value $f(n)$ of some function f ? This is likely to depend on the value of n and, in particular, on how big it is. We will suppose that n is written in binary and has B digits (so $n < 2^B$). Each elementary operation, such as adding two binary bits, or comparing two bits, takes a maximum time τ . So the time to compute $f(n)$ will be at most the time τ multiplied by the number of elementary operations required to do the computation. We will say that f can be *computed in polynomial time* if there is an algorithm to compute $f(n)$ that takes at most time cB^k for all natural numbers n with B bits. Here c and k are some constants independent of n . Similarly, a function of several variables $f(n_1, n_2, \dots, n_r)$ can be computed in polynomial time if there is an algorithm that computes $f(n_1, n_2, \dots, n_r)$ in at most time cB^k where $B = B_1 + B_2 + \dots + B_r$ is the sum of the bit lengths of the numbers n_1, n_2, \dots, n_r .

It is easy to give examples of functions that can be computed in polynomial time.

- (a) Adding or subtracting binary integers.
- (b) Multiplying binary integers.
- (c) Division of binary integers to give a quotient and remainder.
- (d) Computing highest common factors, using Euclid's algorithm.

- (e) Modular arithmetic (addition, subtraction, multiplication, division) modulo N .
- (f) Exponentiation modulo N . To compute α^n , expand n in binary as $\sum n_j 2^j$ with each $n_j = 0$ or 1 . Then compute the powers $\alpha, \alpha^2, \alpha^4, \dots, \alpha^{2^j}, \dots, \alpha^{2^B}$ by repeated squaring and

$$\alpha^n = \prod_{n_j=1} \alpha^{2^j}$$

as a product.

- (g) Testing primality. Agrawal, Kayal and Saxena (2002) gave a polynomial time algorithm to test primality.

However, there are functions for which polynomial time algorithms are not known. For example:

1. **Factoring**

An integer N is known to be the product of two primes p and q . Find these primes.

2. **Discrete Logarithm**

Let α be a primitive root modulo the prime p , so α is a generator of the cyclic group \mathbb{F}_p^\times . Given $x \in \mathbb{F}_p^\times$, find n with $\alpha^n \equiv x \pmod{p}$. We call n the *logarithm of x to base α* and denote it by $\log_\alpha x$.

The elementary methods for computing 1 and 2 take much longer than polynomial time. To factor N we could try dividing by successive primes up to $N^{1/2}$. This takes time $O(N^{1/2}) = O(2^{B/2})$. To compute the discrete logarithm $\log_\alpha x$, set $m = \lceil p^{1/2} \rceil$ and write $n = qm + r$ with $0 \leq q, r < m$. Then

$$\alpha^n \equiv x \pmod{p} \quad \Leftrightarrow \quad (\alpha^m)^q \equiv x\alpha^{-r} \pmod{p} .$$

So compute $(\alpha^m)^q$ and $x\alpha^{-r}$ for all $0 \leq q, r < m$ and see where they agree. This takes time $O(p^{1/2} \log p) = O(2^{B/2} B)$.

Significantly better methods are known but these are still well short of polynomial time. Using number field sieves we can achieve a time

$$O\left(\exp(cB^{1/3}(\log B)^{2/3})\right) .$$

In practice, it seems to take a long time to solve either 1 or 2 for random large values of p , q and x . We will describe various ciphers based on the difficulty of solving these problems.

The RSA cipher relies on factoring being difficult. The RSA laboratories offer rewards for factoring large challenge numbers they give. The RSA- B challenge involves factoring a number with B binary bits. The recent successes were

Challenge Number	Decimal digits	Factored	Prize
RSA-576	174	December, 2003	\$10,000
RSA-640	193	November, 2005	\$20,000
RSA-704	212		\$30,000

(See <http://www.rsa.com/rsalabs/node.asp?id=2093>.)

17.2 Modular Arithmetic

For each natural number N , the integers modulo N will be denoted by \mathbb{Z}_N . The set of integers modulo N that are coprime to N :

$$\mathbb{Z}_N^\times = \{a = 0, 1, 2, \dots, N - 1 : (a, N) = 1\}$$

is a multiplicative group of order $\varphi(N)$ — the *Euler totient function*. Hence Lagrange's theorem shows that

$$a^{\varphi(N)} \equiv 1 \pmod{N} \quad \text{for each } a \text{ with } (a, N) = 1 .$$

This is the *Euler – Fermat theorem*.

When N is a prime p we know that $\varphi(p) = p - 1$, and $\mathbb{Z}_p^\times = \mathbb{F}_p^\times$ consists of all the non-zero elements in the field \mathbb{F}_p . The set $\mathbb{Z}_p^\times = \mathbb{F}_p^\times$ is a cyclic group of order $\varphi(p) = p - 1$. An integer a is a *primitive root modulo p* if a is a generator of this cyclic group. The Euler – Fermat theorem gives Fermat’s little theorem:

$$a^p \equiv a \pmod{p} \quad \text{for all integers } a .$$

When N is a product of two distinct primes, say $N = pq$, then $\varphi(N) = (p - 1)(q - 1)$. Fermat’s little theorem shows that, for each integer k ,

$$a^{k(p-1)+1} \equiv a \pmod{p} \quad \text{for all integers } a .$$

So, in particular, $a^{k\varphi(N)+1} \equiv a \pmod{p}$. Similarly, $a^{k\varphi(N)+1} \equiv a \pmod{q}$. So we obtain

$$a^{k\varphi(N)+1} \equiv a \pmod{N} \quad \text{for all integers } a .$$

Theorem 17.1 Chinese Remainder Theorem

Let p, q be two coprime integers, so there are integers a, b with $ap + bq = 1$.

The equations $x \equiv c \pmod{p}$ and $x \equiv d \pmod{q}$ are equivalent to

$$x \equiv c + ap(d - c) \pmod{pq} .$$

Proof:

If $x = c + ap(d - c) = c + (1 - bq)(d - c) = d - bq(d - c)$, then it is clear that $x \equiv c \pmod{p}$ and $x \equiv d \pmod{q}$.

Conversely, suppose that $x \equiv c \pmod{p}$ and $x \equiv d \pmod{q}$, so $p \mid (x - c)$ and $q \mid (x - d)$. Then

$$p \mid (x - c - ap(d - c)) \quad \text{and} \quad q \mid (x - d - bq(d - c)) .$$

This shows that pq divides $x - d - bq(d - c) = x - c - ap(d - c)$, so

$$x \equiv c + ap(d - c) \pmod{pq} .$$

□

The Chinese Remainder theorem shows that the multiplicative group homomorphism

$$\mathbb{Z}_{pq}^{\times} \rightarrow \mathbb{Z}_p^{\times} \times \mathbb{Z}_q^{\times} ; \quad a \mapsto (a \pmod{p}, a \pmod{q})$$

is an isomorphism. In particular, $\varphi(pq) = \varphi(p)\varphi(q)$ for coprime integers p and q .

Proposition 17.2 Quadratic residues

Let p be a prime. In the field \mathbb{F}_p , the only square root of 0 is 0; precisely $\frac{1}{2}(p - 1)$ elements have no square root and the remaining $\frac{1}{2}(p - 1)$ have exactly two square roots.

Proof:

Since \mathbb{F}_p is a field, we have

$$x^2 \equiv y^2 \pmod{p} \quad \Leftrightarrow \quad (x - y)(x + y) \equiv 0 \pmod{p} .$$

Hence, the squaring map $s : \mathbb{F}_p \rightarrow \mathbb{F}_p; x \mapsto x^2 \pmod{p}$ has $s(x) = s(y)$ if and only if $y = \pm x$. This means that $s^{-1}(0) = \{0\}$ and every other square has two square roots. Hence there must be $\frac{1}{2}(p - 1)$ squares. □

Proposition 17.3 Square roots modulo pq

Let N be the product of two distinct primes p and q . Suppose that the integer a is a square modulo N . Then either:

- (a) $(a, N) = 1$ and there are exactly 4 square roots of a modulo N .
- (b) $(a, N) = p$ or q and there are exactly 2 square roots of a modulo N .
- (c) $a \equiv 0 \pmod{N}$ and 0 is the only square root of 0 modulo N .

Proof:

For an integer x we have

$$x^2 \equiv a \pmod{N} \quad \Leftrightarrow \quad x^2 \equiv a \pmod{p} \quad \text{and} \quad x^2 \equiv a \pmod{q} .$$

When $(a, N) = 1$, we have 2 solutions to $x^2 \equiv a \pmod{p}$. Similarly, there are 2 solutions to $x^2 \equiv a \pmod{q}$. Now the Chinese remainder theorem shows that there are 4 solutions to $x^2 \equiv a \pmod{N}$.

When $p|a$ there is only one solution to $x^2 \equiv a \pmod{p}$. This gives parts (b) and (c). □