In general, if a and b are relatively prime and if $q_1, q_2, \ldots, q_t$ is the sequence of quotients obtained from applying the Euclidean algorithm to a and b as in Figure 1.2 on page 13, then the box has the form

		q_1	q_2	$\cdots$	q_{t-1}	q_t
0	1	P_1	P_2	$\ldots$	P_{t-1}	a
1	0	Q_1	Q_2	$\ldots$	Q_{t-1}	b

The entries in the box are calculated using the initial values

$$P_1 = q_1, \qquad Q_1 = 1, \qquad P_2 = q_2 \cdot P_1 + 1, \qquad Q_2 = q_2 \cdot Q_1,$$

and then, for $i \geq 3$, using the formulas

$$P_i = q_i \cdot P_{i-1} + P_{i-2} \qquad \text{and} \qquad Q_i = q_i \cdot Q_{i-1} + Q_{i-2}.$$

The final four entries in the box satisfy

$$a \cdot Q_{t-1} - b \cdot P_{t-1} = (-1)^t.$$

Multiplying both sides by $(-1)^t$ gives the solution $u = (-1)^t Q_{t-1}$ and $v = (-1)^{t+1} P_{t-1}$ to the equation $au + bv = 1$.

Figure 1.3: Solving $au + bv = 1$ using the Euclidean algorithm

1.3 Modular arithmetic

You may have encountered "clock arithmetic" in grade school, where after you get to 12, the next number is 1. This leads to odd-looking equations such as

$$6 + 9 = 3 \qquad \text{and} \qquad 2 - 3 = 11.$$

These look strange, but they are true using clock arithmetic, since for example 11 o'clock is 3 hours before 2 o'clock. So what we are really doing is first computing $2 - 3 = -1$ and then adding 12 to the answer. Similarly, 9 hours after 6 o'clock is 3 o'clock, since $6 + 9 - 12 = 3$.

The theory of *congruences* is a powerful method in number theory that is based on the simple idea of clock arithmetic.

Definition. Let $m \geq 1$ be an integer. We say that the integers a and b are *congruent modulo* m if their difference $a - b$ is divisible by m. We write

$$a \equiv b \pmod{m}$$

to indicate that a and b are congruent modulo m. The number m is called the *modulus*.

Our clock examples may be written as congruences using the modulus $m = 12$:

$$6 + 9 = 15 \equiv 3 \pmod{12} \qquad \text{and} \qquad 2 - 3 = -1 \equiv 11 \pmod{12}.$$

Example 1.12. We have

$$17 \equiv 7 \pmod{5}, \qquad \text{since 5 divides } 10 = 17 - 7.$$

On the other hand,

$$19 \not\equiv 6 \pmod{11}, \qquad \text{since 11 does not divide } 13 = 19 - 6.$$

Notice that the numbers satisfying

$$a \equiv 0 \pmod{m}$$

are the numbers that are divisible by m, i.e., the multiples of m.

The reason that congruence notation is so useful is that congruences behave much like equalities, as the following proposition indicates.

Proposition 1.13. *Let $m \geq 1$ be an integer.*
(a) *If $a_1 \equiv a_2 \pmod{m}$ and $b_1 \equiv b_2 \pmod{m}$, then*

$$a_1 \pm b_1 \equiv a_2 \pm b_2 \pmod{m} \qquad \text{and} \qquad a_1 \cdot b_1 \equiv a_2 \cdot b_2 \pmod{m}.$$

(b) *Let a be an integer. Then*

$$a \cdot b \equiv 1 \pmod{m} \text{ for some integer } b \text{ if and only if } \gcd(a, m) = 1.$$

If such an integer b exists, then we say that b is the (multiplicative) inverse of a modulo m. (*We say "the" inverse, rather than "an" inverse, because any two inverses are congruent modulo m.*)

Proof. (a) We leave this as an exercise; see Exercise 1.14.
(b) Suppose first that $\gcd(a, m) = 1$. Then Theorem 1.11 tells us that we can find integers u and v satisfying $au + mv = 1$. This means that $au - 1 = -mv$ is divisible by m, so by definition, $au \equiv 1 \pmod{m}$. In other words, we can take $b = u$.

For the other direction, suppose that a has an inverse modulo m, say $a \cdot b \equiv 1 \pmod{m}$. This means that $ab - 1 = cm$ for some integer c. It follows that $\gcd(a, m)$ divides $ab - cm = 1$, so $\gcd(a, m) = 1$. This completes the proof that a has an inverse modulo m if and only if $\gcd(a, m) = 1$. □

Proposition 1.13(b) says that if $\gcd(a, m) = 1$, then there exists an inverse b of a modulo m. This has the curious consequence that the fraction $b^{-1} = 1/b$ then has a meaningful interpretation in the world of integers modulo m.

Example 1.14. We take $m = 5$ and $a = 2$. Clearly $\gcd(2, 5) = 1$, so there exists an inverse to 2 modulo 5. The inverse of 2 modulo 5 is 3, since $2 \cdot 3 \equiv 1 \pmod 5$, so $2^{-1} \equiv 3 \pmod 5$. Similarly $\gcd(4, 15) = 1$ so 4^{-1} exists modulo 15. In fact $4 \cdot 4 \equiv 1 \pmod{15}$ so 4 is its own inverse modulo 15.

We can even work with fractions a/d modulo m as long as the denominator is relatively prime to m. For example, we can compute $5/7$ modulo 11 by first observing that $7 \cdot 8 \equiv 1 \pmod{11}$, so $7^{-1} \equiv 8 \pmod{11}$. Then

$$\frac{5}{7} = 5 \cdot 7^{-1} \equiv 5 \cdot 8 \equiv 40 \equiv 7 \pmod{11}.$$

Remark 1.15. In the preceding examples it was easy to find inverses modulo m by trial and error. However, when m is large, it is more challenging to compute a^{-1} modulo m. Note that we showed that inverses exist by using the extended Euclidean algorithm (Theorem 1.11). In order to actually compute the u and v that appear in the equation $au + mv = \gcd(a, m)$, we can apply the Euclidean algorithm directly as we did in Example 1.10, or we can use the somewhat more efficient box method described at the end of the preceding section, or we can use the algorithm given in Exercise 1.12. In any case, since the Euclidean algorithm takes only $2\log_2(b) + 3$ iterations to compute $\gcd(a, b)$, it takes only a small multiple of $\log_2(m)$ steps to compute a^{-1} modulo m.

We now continue our development of the theory of modular arithmetic. If a divided by m has quotient q and remainder r, it can be written as

$$a = m \cdot q + r \qquad \text{with } 0 \le r < m.$$

This shows that $a \equiv r \pmod m$ for some integer r between 0 and $m - 1$, so if we want to work with integers modulo m, it is enough to use the integers $0 \le r < m$. This prompts the following definition.

Definition. We write

$$\mathbb{Z}/m\mathbb{Z} = \{0, 1, 2, \ldots, m - 1\}$$

and call $\mathbb{Z}/m\mathbb{Z}$ the *ring of integers modulo* m. Note that whenever we perform an addition or multiplication in $\mathbb{Z}/m\mathbb{Z}$, we always divide the result by m and take the remainder in order to obtain an element in $\mathbb{Z}/m\mathbb{Z}$.

Figure 1.4 illustrates the ring $\mathbb{Z}/5\mathbb{Z}$ by giving complete addition and multiplication tables modulo 5.

Remark 1.16. If you have studied ring theory, you will recognize that $\mathbb{Z}/m\mathbb{Z}$ is the quotient ring of $\mathbb{Z}$ by the principal ideal $m\mathbb{Z}$, and that the numbers $0, 1, \ldots, m - 1$ are actually coset representatives for the congruence classes that comprise the elements of $\mathbb{Z}/m\mathbb{Z}$. For a discussion of congruence classes and general quotient rings, see Section 2.10.2.

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

·	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Figure 1.4: Addition and multiplication tables modulo 5

Definition. Proposition 1.13(b) tells us that a has an inverse modulo m if and only if $\gcd(a, m) = 1$. Numbers that have inverses are called *units*. We denote the set of all units by

$$\begin{aligned}(\mathbb{Z}/m\mathbb{Z})^* &= \{a \in \mathbb{Z}/m\mathbb{Z} : \gcd(a, m) = 1\} \\ &= \{a \in \mathbb{Z}/m\mathbb{Z} : a \text{ has an inverse modulo } m\}.\end{aligned}$$

The set $(\mathbb{Z}/m\mathbb{Z})^*$ is called the *group of units modulo* m.

Notice that if a_1 and a_2 are units modulo m, then so is a_1a_2. (Do you see why this is true?) So when we multiply two units, we always get a unit. On the other hand, if we add two units, we often do not get a unit.

Example 1.17. The group of units modulo 24 is

$$(\mathbb{Z}/24\mathbb{Z})^* = \{1, 5, 7, 11, 13, 17, 19, 23\}.$$

The multiplication table for $(\mathbb{Z}/24\mathbb{Z})^*$ is illustrated in Figure 1.5.

Example 1.18. The group of units modulo 7 is

$$(\mathbb{Z}/7\mathbb{Z})^* = \{1, 2, 3, 4, 5, 6\},$$

since every number between 1 and 6 is relatively prime to 7. The multiplication table for $(\mathbb{Z}/7\mathbb{Z})^*$ is illustrated in Figure 1.5.

In many of the cryptosystems that we will study, it is important to know how many elements are in the unit group modulo m. This quantity is sufficiently ubiquitous that we give it a name.

Definition. *Euler's phi function* (also sometimes known as *Euler's totient function*) is the function $\phi(m)$ defined by the rule

$$\phi(m) = \#\left(\mathbb{Z}/m\mathbb{Z}\right)^* = \#\{0 \le a < m : \gcd(a, m) = 1\}.$$

For example, we see from Examples 1.17 and 1.18 that $\phi(24) = 8$ and $\phi(7) = 6$.

·	1	5	7	11	13	17	19	23
1	1	5	7	11	13	17	19	23
5	5	1	11	7	17	13	23	19
7	7	11	1	5	19	23	13	17
11	11	7	5	1	23	19	17	13
13	13	17	19	23	1	5	7	11
17	17	13	23	19	5	1	11	7
19	19	23	13	17	7	11	1	5
23	23	19	17	13	11	7	5	1

Unit group modulo 24

·	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

Unit group modulo 7

Figure 1.5: The unit groups $(\mathbb{Z}/24\mathbb{Z})^*$ and $(\mathbb{Z}/7\mathbb{Z})^*$

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Table 1.7: Assigning numbers to letters

1.3.1 Modular arithmetic and shift ciphers

Recall that the Caesar (or shift) cipher studied in Section 1.1 works by shifting each letter in the alphabet a fixed number of letters. We can describe a shift cipher mathematically by assigning a number to each letter as in Table 1.7.

Then a shift cipher with shift k takes a plaintext letter corresponding to the number p and assigns it to the ciphertext letter corresponding to the number $p + k \bmod 26$. Notice how the use of modular arithmetic, in this case modulo 26, simplifies the description of the shift cipher. The shift amount serves as both the encryption key and the decryption key. Encryption is given by the formula

$$(\text{Ciphertext Letter}) \equiv (\text{Plaintext Letter}) + (\text{Secret Key}) \pmod{26},$$

and decryption works by shifting in the opposite direction,

$$(\text{Plaintext Letter}) \equiv (\text{Ciphertext Letter}) - (\text{Secret Key}) \pmod{26}.$$

More succinctly, if we let

$$p = \text{Plaintext Letter}, \qquad c = \text{Ciphertext Letter}, \qquad k = \text{Secret Key},$$

then

$$\underbrace{c \equiv p + k \pmod{26}}_{\text{Encryption}} \qquad \text{and} \qquad \underbrace{p \equiv c - k \pmod{26}}_{\text{Decryption}}.$$

1.3.2 The fast powering algorithm

In some cryptosystems that we will study, for example the RSA and Diffie–Hellman cryptosystems, Alice and Bob are required to compute large powers of a number g modulo another number N, where N may have hundreds of digits. The naive way to compute g^A is by repeated multiplication by g. Thus

$$g_1 \equiv g \pmod{N}, \quad g_2 \equiv g \cdot g_1 \pmod{N}, \quad g_3 \equiv g \cdot g_2 \pmod{N},$$
$$g_4 \equiv g \cdot g_3 \pmod{N}, \quad g_5 \equiv g \cdot g_4 \pmod{N}, \ldots.$$

It is clear that $g_A \equiv g^A \pmod{N}$, but if A is large, this algorithm is completely impractical. For example, if $A \approx 2^{1000}$, then the naive algorithm would take longer than the estimated age of the universe! Clearly if it is to be useful, we need to find a better way to compute $g^A \pmod{N}$.

The idea is to use the binary expansion of the exponent A to convert the calculation of g^A into a succession of squarings and multiplications. An example will make the idea clear, after which we give a formal description of the method.

Example 1.19. Suppose that we want to compute $3^{218} \pmod{1000}$. The first step is to write 218 as a sum of powers of 2,

$$218 = 2 + 2^3 + 2^4 + 2^6 + 2^7.$$

Then 3^{218} becomes

$$3^{218} = 3^{2+2^3+2^4+2^6+2^7} = 3^2 \cdot 3^{2^3} \cdot 3^{2^4} \cdot 3^{2^6} \cdot 3^{2^7}. \tag{1.3}$$

Notice that it is relatively easy to compute the sequence of values

$$3, \quad 3^2, \quad 3^{2^2}, \quad 3^{2^3}, \quad 3^{2^4}, \ldots,$$

since each number in the sequence is the square of the preceding one. Further, since we only need these values modulo 1000, we never need to store more than three digits. Table 1.8 lists the powers of 3 modulo 1000 up to 3^{2^7}. Creating Table 1.8 requires only 7 multiplications, despite the fact that the number $3^{2^7} = 3^{128}$ has quite a large exponent, because each successive entry in the table is equal to the square of the previous entry.

We use (1.3) to decide which powers from Table 1.8 are needed to compute 3^{218}. Thus

i	0	1	2	3	4	5	6	7
$3^{2^i} \pmod{1000}$	3	9	81	561	721	841	281	961

Table 1.8: Successive square powers of 3 modulo 1000

$$\begin{aligned} 3^{218} &= 3^2 \cdot 3^{2^3} \cdot 3^{2^4} \cdot 3^{2^6} \cdot 3^{2^7} \\ &\equiv 9 \cdot 561 \cdot 721 \cdot 281 \cdot 961 \pmod{1000} \\ &\equiv 489 \pmod{1000}. \end{aligned}$$

We note that in computing the product $9 \cdot 561 \cdot 721 \cdot 281 \cdot 961$, we may reduce modulo 1000 after each multiplication, so we never need to deal with very large numbers. We also observe that it has taken us only 11 multiplications to compute $3^{218} \pmod{1000}$, a huge savings over the naive approach. And for larger exponents we would save even more.

The general approach used in Example 1.19 goes by various names, including the *Fast Powering Algorithm* and the *Square-and-Multiply Algorithm*. We now describe the algorithm more formally.

The Fast Powering Algorithm

Step 1. Compute the binary expansion of A as

$$A = A_0 + A_1 \cdot 2 + A_2 \cdot 2^2 + A_3 \cdot 2^3 + \cdots + A_r \cdot 2^r \quad \text{with } A_0, \ldots, A_r \in \{0, 1\},$$

where we may assume that $A_r = 1$.

Step 2. Compute the powers $g^{2^i} \pmod{N}$ for $0 \le i \le r$ by successive squaring,

$$\begin{aligned} a_0 &\equiv g && \pmod{N} \\ a_1 &\equiv a_0^2 \equiv g^2 && \pmod{N} \\ a_2 &\equiv a_1^2 \equiv g^{2^2} && \pmod{N} \\ a_3 &\equiv a_2^2 \equiv g^{2^3} && \pmod{N} \\ &\vdots \quad \vdots && \vdots \\ a_r &\equiv a_{r-1}^2 \equiv g^{2^r} && \pmod{N}. \end{aligned}$$

Each term is the square of the previous one, so this requires r multiplications.

Step 3. Compute $g^A \pmod{N}$ using the formula

$$\begin{aligned} g^A &= g^{A_0+A_1\cdot 2+A_2\cdot 2^2+A_3\cdot 2^3+\cdots+A_r\cdot 2^r} \\ &= g^{A_0}\cdot (g^2)^{A_1}\cdot (g^{2^2})^{A_2}\cdot (g^{2^3})^{A_3}\cdots (g^{2^r})^{A_r} \\ &\equiv a_0^{A_0}\cdot a_1^{A_1}\cdot a_2^{A_2}\cdot a_3^{A_3}\cdots a_r^{A_r} \pmod N. \end{aligned} \tag{1.4}$$

Note that the quantities $a_0, a_1, \ldots, a_r$ were computed in Step 2. Thus the product (1.4) can be computed by looking up the values of the a_i's whose exponent A_i is 1 and then multiplying them together. This requires at most another r multiplications.

Running Time. It takes at most $2r$ multiplications modulo N to compute g^A. Since $A \geq 2^r$, we see that it takes at most $2\log_2(A)$ multiplications[8] modulo N to compute g^A. Thus even if A is very large, say $A \approx 2^{1000}$, it is easy for a computer to do the approximately 2000 multiplications needed to calculate 2^A modulo N.

Efficiency Issues. There are various ways in which the square-and-multiply algorithm can be made somewhat more efficient, in particular regarding eliminating storage requirements; see Exercise 1.24 for an example.

1.4 Prime numbers, unique factorization, and finite fields

In Section 1.3 we studied modular arithmetic and saw that it makes sense to add, subtract, and multiply integers modulo m. Division, however, can be problematic, since we can divide by a in $\mathbb{Z}/m\mathbb{Z}$ only if $\gcd(a, m) = 1$. But notice that if the integer m is a prime, then we can divide by every nonzero element of $\mathbb{Z}/m\mathbb{Z}$. We start with a brief discussion of prime numbers before returning to the ring $\mathbb{Z}/p\mathbb{Z}$ with p prime.

Definition. An integer p is called a *prime* if $p \geq 2$ and if the only positive integers dividing p are 1 and p.

For example, the first ten primes are $2, 3, 5, 7, 11, 13, 17, 19, 23, 29$, while the hundred thousandth prime is 1299709 and the millionth is 15485863. There are infinitely many primes, a fact that was known in ancient Greece and appears as a theorem in Euclid's *Elements*. (See Exercise 1.26.)

A prime p is defined in terms of the numbers that divide p. So the following proposition, which describes a useful property of numbers that are divisible by p, is not obvious and needs to be carefully proved. Notice that the proposition is false for composite numbers. For example, 6 divides $3\cdot 10$, but 6 divides neither 3 nor 10.

[8] Note that $\log_2(A)$ means the usual logarithm to the base 2, not the so-called discrete logarithm that will be discussed in Chapter 2.