31.5-2
Find all integers \( x \) that leave remainders 1, 2, 3 when divided by 9, 8, 7 respectively.

31.5-3
Argue that, under the definitions of Theorem 31.27, if \( \gcd(a, n) = 1 \), then
\[
(a^{-1} \mod n) \leftrightarrow ((a^{-1}_1 \mod n_1), (a^{-1}_2 \mod n_2), \ldots, (a^{-1}_k \mod n_k)) \, .
\]

31.5-4
Under the definitions of Theorem 31.27, prove that for any polynomial \( f \), the number of roots of the equation \( f(x) \equiv 0 \) (mod \( n \)) equals the product of the number of roots of each of the equations \( f(x) \equiv 0 \) (mod \( n_1 \), \( f(x) \equiv 0 \) (mod \( n_2 \), \ldots, \( f(x) \equiv 0 \) (mod \( n_k \)).

31.6 Powers of an element

Just as we often consider the multiples of a given element \( a \), modulo \( n \), we consider the sequence of powers of \( a \), modulo \( n \), where \( a \in \mathbb{Z}_n^* \):
\[
a^0, a^1, a^2, a^3, \ldots, \tag{31.33}
\]
modulo \( n \). Indexing from 0, the 0th value in this sequence is \( a^0 \bmod n = 1 \), and the \( i \)th value is \( a^i \bmod n \). For example, the powers of 3 modulo 7 are

\[
i \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11 \quad \ldots
\]
\[
3^i \bmod 7 \quad 1 \quad 3 \quad 2 \quad 6 \quad 4 \quad 5 \quad 1 \quad 3 \quad 2 \quad 6 \quad 4 \quad 5 \quad \ldots
\]
whereas the powers of 2 modulo 7 are

\[
i \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11 \quad \ldots
\]
\[
2^i \bmod 7 \quad 1 \quad 2 \quad 4 \quad 1 \quad 2 \quad 4 \quad 1 \quad 2 \quad 4 \quad 1 \quad 2 \quad 4 \quad \ldots
\]

In this section, let \( \langle a \rangle \) denote the subgroup of \( \mathbb{Z}_n^* \) generated by \( a \) by repeated multiplication, and let \( \text{ord}_n(a) \) (the “order of \( a \), modulo \( n \)”) denote the order of \( a \) in \( \mathbb{Z}_n^* \). For example, \( \langle 2 \rangle = \{1, 2, 4\} \) in \( \mathbb{Z}_7^* \), and \( \text{ord}_7(2) = 3 \). Using the definition of the Euler phi function \( \phi(n) \) as the size of \( \mathbb{Z}_n^* \) (see Section 31.3), we now translate Corollary 31.19 into the notation of \( \mathbb{Z}_n^* \) to obtain Euler’s theorem and specialize it to \( \mathbb{Z}_p^* \), where \( p \) is prime, to obtain Fermat’s theorem.

**Theorem 31.30 (Euler’s theorem)**

For any integer \( n > 1 \),

\[
a^{\phi(n)} \equiv 1 \pmod{n} \quad \text{for all } a \in \mathbb{Z}_n^* .
\]
31.6 Powers of an element

**Theorem 31.31 (Fermat’s theorem)**

If $p$ is prime, then

$$a^{p-1} \equiv 1 \pmod{p} \text{ for all } a \in \mathbb{Z}_p^*.$$

**Proof** By equation (31.21), $\phi(p) = p - 1$ if $p$ is prime.

Fermat’s theorem applies to every element in $\mathbb{Z}_p$ except 0, since 0 $\notin \mathbb{Z}_p^*$. For all $a \in \mathbb{Z}_p$, however, we have $a^p \equiv a \pmod{p}$ if $p$ is prime.

If $\text{ord}_n(g) = |\mathbb{Z}_n^*|$, then every element in $\mathbb{Z}_n^*$ is a power of $g$, modulo $n$, and $g$ is a **primitive root** or a **generator** of $\mathbb{Z}_n^*$. For example, 3 is a primitive root, modulo 7, but 2 is not a primitive root, modulo 7. If $\mathbb{Z}_n^*$ possesses a primitive root, the group $\mathbb{Z}_n^*$ is **cyclic**. We omit the proof of the following theorem, which is proven by Niven and Zuckerman [265].

**Theorem 31.32**

The values of $n > 1$ for which $\mathbb{Z}_n^*$ is cyclic are 2, 4, $p^e$, and $2p^e$, for all primes $p > 2$ and all positive integers $e$.

If $g$ is a primitive root of $\mathbb{Z}_n^*$ and $a$ is any element of $\mathbb{Z}_n^*$, then there exists a $z$ such that $g^z \equiv a \pmod{n}$. This $z$ is a **discrete logarithm** or an **index** of $a$, modulo $n$, to the base $g$; we denote this value as $\text{ind}_{n,g}(a)$.

**Theorem 31.33 (Discrete logarithm theorem)**

If $g$ is a primitive root of $\mathbb{Z}_n^*$, then the equation $g^x \equiv g^y \pmod{n}$ holds if and only if the equation $x \equiv y \pmod{\phi(n)}$ holds.

**Proof** Suppose first that $x \equiv y \pmod{\phi(n)}$. Then, $x = y + k\phi(n)$ for some integer $k$. Therefore,

$$g^x \equiv g^{y+k\phi(n)} \pmod{n} \equiv g^y \cdot (g^{\phi(n)})^k \pmod{n} \equiv g^y \cdot 1^k \pmod{n} \quad \text{(by Euler’s theorem)} \equiv g^y \pmod{n}.$$  

Conversely, suppose that $g^x \equiv g^y \pmod{n}$. Because the sequence of powers of $g$ generates every element of $\langle g \rangle$ and $|\langle g \rangle| = \phi(n)$, Corollary 31.18 implies that the sequence of powers of $g$ is periodic with period $\phi(n)$. Therefore, if $g^x \equiv g^y \pmod{n}$, then we must have $x \equiv y \pmod{\phi(n)}$.

We now turn our attention to the square roots of 1, modulo a prime power. The following theorem will be useful in our development of a primality-testing algorithm in Section 31.8.
**Theorem 31.34**

If $p$ is an odd prime and $e \geq 1$, then the equation

$$x^2 \equiv 1 \pmod{p^e}$$

(31.34)

has only two solutions, namely $x = 1$ and $x = -1$.

**Proof** Equation (31.34) is equivalent to

$$p^e \mid (x - 1)(x + 1).$$

Since $p > 2$, we can have $p \mid (x - 1)$ or $p \mid (x + 1)$, but not both. (Otherwise, by property (31.3), $p$ would also divide their difference $x + 1 - (x - 1) = 2$.) If $p \nmid (x - 1)$, then gcd$(p^e, x - 1) = 1$, and by Corollary 31.5, we would have $p^e \mid (x + 1)$. That is, $x \equiv -1 \pmod{p^e}$. Symmetrically, if $p \nmid (x + 1)$, then gcd$(p^e, x + 1) = 1$, and Corollary 31.5 implies that $p^e \mid (x - 1)$, so that $x \equiv 1 \pmod{p^e}$. Therefore, either $x \equiv -1 \pmod{p^e}$ or $x \equiv 1 \pmod{p^e}$.

An integer $x$ is a **nontrivial square root of 1, modulo $n$**, if it satisfies the equation $x^2 \equiv 1 \pmod{n}$ but $x$ is equivalent to neither of the two “trivial” square roots: 1 or $-1$, modulo $n$. For example, 6 is a nontrivial square root of 1, modulo 35. We shall use the following corollary to Theorem 31.34 in the correctness proof in Section 31.8 for the Miller-Rabin primality-testing procedure.

**Corollary 31.35**

If there exists a nontrivial square root of 1, modulo $n$, then $n$ is composite.

**Proof** By the contrapositive of Theorem 31.34, if there exists a nontrivial square root of 1, modulo $n$, then $n$ cannot be an odd prime or a power of an odd prime. If $x^2 \equiv 1 \pmod{2}$, then $x \equiv 1 \pmod{2}$, and so all square roots of 1, modulo 2, are trivial. Thus, $n$ cannot be prime. Finally, we must have $n > 1$ for a nontrivial square root of 1 to exist. Therefore, $n$ must be composite.

**Raising to powers with repeated squaring**

A frequently occurring operation in number-theoretic computations is raising one number to a power modulo another number, also known as **modular exponentiation**. More precisely, we would like an efficient way to compute $a^b \pmod{n}$, where $a$ and $b$ are nonnegative integers and $n$ is a positive integer. Modular exponentiation is an essential operation in many primality-testing routines and in the RSA public-key cryptosystem. The method of **repeated squaring** solves this problem efficiently using the binary representation of $b$.

Let $(b_k, b_{k-1}, \ldots, b_1, b_0)$ be the binary representation of $b$. (That is, the binary representation is $k + 1$ bits long, $b_k$ is the most significant bit, and $b_0$ is the least
31.6 Powers of an element 957

\[ \begin{array}{cccccccccccc}
  i & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
  b_i & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
  c & 7 & 49 & 157 & 526 & 160 & 241 & 298 & 166 & 67 & 1 \\
  d & 1 & 2 & 4 & 8 & 17 & 35 & 70 & 140 & 280 & 560 \\
\end{array} \]

**Figure 31.4** The results of MODULAR-EXPONENTIATION when computing \( a^b \pmod{n} \), where \( a = 7 \), \( b = 560 = (1000110000) \), and \( n = 561 \). The values are shown after each execution of the for loop. The final result is 1.

significant bit.) The following procedure computes \( a^c \pmod{n} \) as \( c \) is increased by doublings and incrementations from 0 to \( b \).

**MODULAR-EXPONENTIATION**(\( a, b, n \))

1. \( c = 0 \)
2. \( d = 1 \)
3. let \( \langle b_k, b_{k-1}, \ldots, b_0 \rangle \) be the binary representation of \( b \)
4. for \( i = k \) downto 0
   5. \( c = 2c \)
   6. \( d = (d \cdot d) \pmod{n} \)
   7. if \( b_i = 1 \)
      8. \( c = c + 1 \)
   9. \( d = (d \cdot a) \pmod{n} \)
10. return \( d \)

The essential use of squaring in line 6 of each iteration explains the name “repeated squaring.” As an example, for \( a = 7 \), \( b = 560 \), and \( n = 561 \), the algorithm computes the sequence of values modulo 561 shown in Figure 31.4; the sequence of exponents used appears in the row of the table labeled by \( c \).

The variable \( c \) is not really needed by the algorithm but is included for the following two-part loop invariant:

Just prior to each iteration of the for loop of lines 4–9,

1. The value of \( c \) is the same as the prefix \( \langle b_k, b_{k-1}, \ldots, b_{i+1} \rangle \) of the binary representation of \( b \), and
2. \( d = a^c \pmod{n} \).

We use this loop invariant as follows:

**Initialization:** Initially, \( i = k \), so that the prefix \( \langle b_k, b_{k-1}, \ldots, b_{i+1} \rangle \) is empty, which corresponds to \( c = 0 \). Moreover, \( d = 1 = a^0 \pmod{n} \).
**Maintenance:** Let \( c' \) and \( d' \) denote the values of \( c \) and \( d \) at the end of an iteration of the for loop, and thus the values prior to the next iteration. Each iteration updates \( c' = 2c \) (if \( b_i = 0 \)) or \( c' = 2c + 1 \) (if \( b_i = 1 \)), so that \( c \) will be correct prior to the next iteration. If \( b_i = 0 \), then \( d' = d^2 \mod n = (a^c)^2 \mod n = a^{2c} \mod n = a^c \mod n \). If \( b_i = 1 \), then \( d' = d^2a \mod n = (a^c)^2a \mod n = a^{2c+1} \mod n = a^{c'} \mod n \). In either case, \( d = a^c \mod n \) prior to the next iteration.

**Termination:** At termination, \( i = -1 \). Thus, \( c = b \), since \( c \) has the value of the prefix \( \langle b_k, b_{k-1}, \ldots, b_0 \rangle \) of \( b \)’s binary representation. Hence \( d = a^c \mod n = a^b \mod n \).

If the inputs \( a, b, \) and \( n \) are \( \beta \)-bit numbers, then the total number of arithmetic operations required is \( O(\beta) \) and the total number of bit operations required is \( O(\beta^3) \).

**Exercises**

**31.6-1**
Draw a table showing the order of every element in \( \mathbb{Z}_{11}^* \). Pick the smallest primitive root \( g \) and compute a table giving \( \text{ind}_{11, g}(x) \) for all \( x \in \mathbb{Z}_{11}^* \).

**31.6-2**
Give a modular exponentiation algorithm that examines the bits of \( b \) from right to left instead of left to right.

**31.6-3**
Assuming that you know \( \phi(n) \), explain how to compute \( a^{-1} \mod n \) for any \( a \in \mathbb{Z}_n^* \) using the procedure \textsc{Modular-Exponentiation}.

**31.7 The RSA public-key cryptosystem**

With a public-key cryptosystem, we can encrypt messages sent between two communicating parties so that an eavesdropper who overhears the encrypted messages will not be able to decode them. A public-key cryptosystem also enables a party to append an unforgeable “digital signature” to the end of an electronic message. Such a signature is the electronic version of a handwritten signature on a paper document. It can be easily checked by anyone, forged by no one, yet loses its validity if any bit of the message is altered. It therefore provides authentication of both the identity of the signer and the contents of the signed message. It is the perfect tool