we won't prove it here. The reason is that for a randomly chosen odd composite integer n, the expected number of nonwitnesses to the compositeness of n is likely to be very much smaller than $(n-1)/2$.

If the integer n is not chosen randomly, however, the best that can be proven is that the number of nonwitnesses is at most $(n-1)/4$, using an improved version of Theorem 31.38. Furthermore, there do exist integers n for which the number of nonwitnesses is $(n-1)/4$.

Exercises

31.8-1

Prove that if an odd integer $n > 1$ is not a prime or a prime power, then there exists a nontrivial square root of 1 modulo n.

31.8-2 ⋆

It is possible to strengthen Euler's theorem slightly to the form

$$a^{\lambda(n)} \equiv 1 \pmod{n} \text{ for all } a \in \mathbb{Z}_n^* ,$$

where $n = p_1^{e_1} \cdots p_r^{e_r}$ and $\lambda(n)$ is defined by

$$\lambda(n) = \text{lcm}(\phi(p_1^{e_1}), \ldots, \phi(p_r^{e_r})) . \tag{31.42}$$

Prove that $\lambda(n) \mid \phi(n)$. A composite number n is a Carmichael number if $\lambda(n) \mid n-1$. The smallest Carmichael number is $561 = 3 \cdot 11 \cdot 17$; here, $\lambda(n) = \text{lcm}(2, 10, 16) = 80$, which divides 560. Prove that Carmichael numbers must be both "square-free" (not divisible by the square of any prime) and the product of at least three primes. (For this reason, they are not very common.)

31.8-3

Prove that if x is a nontrivial square root of 1, modulo n, then $\gcd(x-1, n)$ and $\gcd(x+1, n)$ are both nontrivial divisors of n.

⋆ 31.9 Integer factorization

Suppose we have an integer n that we wish to ***factor***, that is, to decompose into a product of primes. The primality test of the preceding section may tell us that n is composite, but it does not tell us the prime factors of n. Factoring a large integer n seems to be much more difficult than simply determining whether n is prime or composite. Even with today's supercomputers and the best algorithms to date, we cannot feasibly factor an arbitrary 1024-bit number.

Pollard's rho heuristic

Trial division by all integers up to R is guaranteed to factor completely any number up to R^2. For the same amount of work, the following procedure, POLLARD-RHO, factors any number up to R^4 (unless we are unlucky). Since the procedure is only a heuristic, neither its running time nor its success is guaranteed, although the procedure is highly effective in practice. Another advantage of the POLLARD-RHO procedure is that it uses only a constant number of memory locations. (If you wanted to, you could easily implement POLLARD-RHO on a programmable pocket calculator to find factors of small numbers.)

```
POLLARD-RHO($n$)
 1  $i = 1$
 2  $x_1 = \text{RANDOM}(0, n-1)$
 3  $y = x_1$
 4  $k = 2$
 5  while TRUE
 6      $i = i + 1$
 7      $x_i = (x_{i-1}^2 - 1) \bmod n$
 8      $d = \gcd(y - x_i, n)$
 9      if $d \neq 1$ and $d \neq n$
10          print $d$
11      if $i == k$
12          $y = x_i$
13          $k = 2k$
```

The procedure works as follows. Lines 1–2 initialize i to 1 and x_1 to a randomly chosen value in $\mathbb{Z}_n$. The **while** loop beginning on line 5 iterates forever, searching for factors of n. During each iteration of the **while** loop, line 7 uses the recurrence

$$x_i = (x_{i-1}^2 - 1) \bmod n \tag{31.43}$$

to produce the next value of x_i in the infinite sequence

$$x_1, x_2, x_3, x_4, \ldots , \tag{31.44}$$

with line 6 correspondingly incrementing i. The pseudocode is written using subscripted variables x_i for clarity, but the program works the same if all of the subscripts are dropped, since only the most recent value of x_i needs to be maintained. With this modification, the procedure uses only a constant number of memory locations.

Every so often, the program saves the most recently generated x_i value in the variable y. Specifically, the values that are saved are the ones whose subscripts are powers of 2:

$x_1, x_2, x_4, x_8, x_{16}, \ldots$.

Line 3 saves the value x_1, and line 12 saves x_k whenever i is equal to k. The variable k is initialized to 2 in line 4, and line 13 doubles it whenever line 12 updates y. Therefore, k follows the sequence $2, 4, 8, 16 \ldots$ and always gives the subscript of the next value x_k to be saved in y.

Lines 8–10 try to find a factor of n, using the saved value of y and the current value of x_i. Specifically, line 8 computes the greatest common divisor $d = \gcd(y - x_i, n)$. If line 9 finds d to be a nontrivial divisor of n, then line 10 prints d.

This procedure for finding a factor may seem somewhat mysterious at first. Note, however, that POLLARD-RHO never prints an incorrect answer; any number it prints is a nontrivial divisor of n. POLLARD-RHO might not print anything at all, though; it comes with no guarantee that it will print any divisors. We shall see, however, that we have good reason to expect POLLARD-RHO to print a factor p of n after $\Theta(\sqrt{p})$ iterations of the **while** loop. Thus, if n is composite, we can expect this procedure to discover enough divisors to factor n completely after approximately $n^{1/4}$ updates, since every prime factor p of n except possibly the largest one is less than $\sqrt{n}$.

We begin our analysis of how this procedure behaves by studying how long it takes a random sequence modulo n to repeat a value. Since $\mathbb{Z}_n$ is finite, and since each value in the sequence (31.44) depends only on the previous value, the sequence (31.44) eventually repeats itself. Once we reach an x_i such that $x_i = x_j$ for some $j < i$, we are in a cycle, since $x_{i+1} = x_{j+1}$, $x_{i+2} = x_{j+2}$, and so on. The reason for the name "rho heuristic" is that, as Figure 31.7 shows, we can draw the sequence $x_1, x_2, \ldots, x_{j-1}$ as the "tail" of the rho and the cycle $x_j, x_{j+1}, \ldots, x_i$ as the "body" of the rho.

Let us consider the question of how long it takes for the sequence of x_i to repeat. This information is not exactly what we need, but we shall see later how to modify the argument. For the purpose of this estimation, let us assume that the function

$$f_n(x) = (x^2 - 1) \bmod n$$

behaves like a "random" function. Of course, it is not really random, but this assumption yields results consistent with the observed behavior of POLLARD-RHO. We can then consider each x_i to have been independently drawn from $\mathbb{Z}_n$ according to a uniform distribution on $\mathbb{Z}_n$. By the birthday-paradox analysis of Section 5.4.1, we expect $\Theta(\sqrt{n})$ steps to be taken before the sequence cycles.

Now for the required modification. Let p be a nontrivial factor of n such that $\gcd(p, n/p) = 1$. For example, if n has the factorization $n = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$, then we may take p to be $p_1^{e_1}$. (If $e_1 = 1$, then p is just the smallest prime factor of n, a good example to keep in mind.)

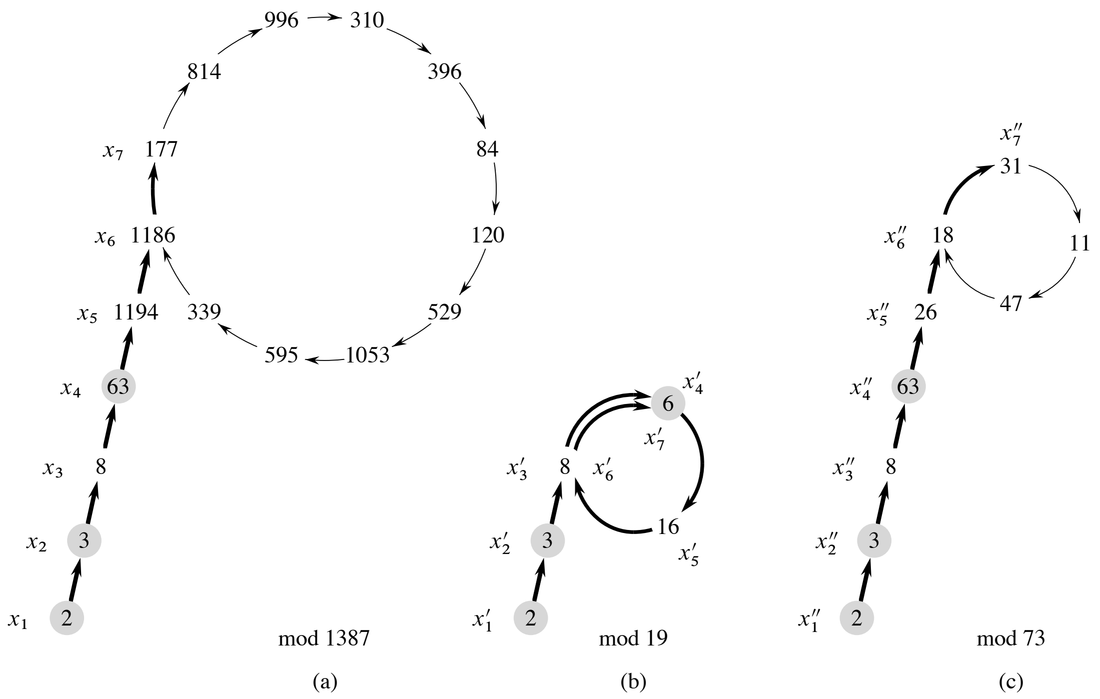

Figure 31.7 Pollard's rho heuristic. **(a)** The values produced by the recurrence $x_{i+1} = (x_i^2 - 1) \bmod 1387$, starting with $x_1 = 2$. The prime factorization of 1387 is $19 \cdot 73$. The heavy arrows indicate the iteration steps that are executed before the factor 19 is discovered. The light arrows point to unreached values in the iteration, to illustrate the "rho" shape. The shaded values are the y values stored by POLLARD-RHO. The factor 19 is discovered upon reaching $x_7 = 177$, when $\gcd(63 - 177, 1387) = 19$ is computed. The first x value that would be repeated is 1186, but the factor 19 is discovered before this value is repeated. **(b)** The values produced by the same recurrence, modulo 19. Every value x_i given in part (a) is equivalent, modulo 19, to the value x'_i shown here. For example, both $x_4 = 63$ and $x_7 = 177$ are equivalent to 6, modulo 19. **(c)** The values produced by the same recurrence, modulo 73. Every value x_i given in part (a) is equivalent, modulo 73, to the value x''_i shown here. By the Chinese remainder theorem, each node in part (a) corresponds to a pair of nodes, one from part (b) and one from part (c).

The sequence $\langle x_i \rangle$ induces a corresponding sequence $\langle x'_i \rangle$ modulo p, where

$$x'_i = x_i \bmod p$$

for all i.

Furthermore, because f_n is defined using only arithmetic operations (squaring and subtraction) modulo n, we can compute x'_{i+1} from x'_i; the "modulo p" view of the sequence is a smaller version of what is happening modulo n:

$$\begin{aligned}
x'_{i+1} &= x_{i+1} \bmod p \\
&= f_n(x_i) \bmod p \\
&= ((x_i^2 - 1) \bmod n) \bmod p \\
&= (x_i^2 - 1) \bmod p \qquad \text{(by Exercise 31.1-7)} \\
&= ((x_i \bmod p)^2 - 1) \bmod p \\
&= ((x'_i)^2 - 1) \bmod p \\
&= f_p(x'_i) .
\end{aligned}$$

Thus, although we are not explicitly computing the sequence $\langle x'_i \rangle$, this sequence is well defined and obeys the same recurrence as the sequence $\langle x_i \rangle$.

Reasoning as before, we find that the expected number of steps before the sequence $\langle x'_i \rangle$ repeats is $\Theta(\sqrt{p})$. If p is small compared to n, the sequence $\langle x'_i \rangle$ might repeat much more quickly than the sequence $\langle x_i \rangle$. Indeed, as parts (b) and (c) of Figure 31.7 show, the $\langle x'_i \rangle$ sequence repeats as soon as two elements of the sequence $\langle x_i \rangle$ are merely equivalent modulo p, rather than equivalent modulo n.

Let t denote the index of the first repeated value in the $\langle x'_i \rangle$ sequence, and let $u > 0$ denote the length of the cycle that has been thereby produced. That is, t and $u > 0$ are the smallest values such that $x'_{t+i} = x'_{t+u+i}$ for all $i \geq 0$. By the above arguments, the expected values of t and u are both $\Theta(\sqrt{p})$. Note that if $x'_{t+i} = x'_{t+u+i}$, then $p \mid (x_{t+u+i} - x_{t+i})$. Thus, $\gcd(x_{t+u+i} - x_{t+i}, n) > 1$.

Therefore, once POLLARD-RHO has saved as y any value x_k such that $k \geq t$, then $y \bmod p$ is always on the cycle modulo p. (If a new value is saved as y, that value is also on the cycle modulo p.) Eventually, k is set to a value that is greater than u, and the procedure then makes an entire loop around the cycle modulo p without changing the value of y. The procedure then discovers a factor of n when x_i "runs into" the previously stored value of y, modulo p, that is, when $x_i \equiv y \pmod{p}$.

Presumably, the factor found is the factor p, although it may occasionally happen that a multiple of p is discovered. Since the expected values of both t and u are $\Theta(\sqrt{p})$, the expected number of steps required to produce the factor p is $\Theta(\sqrt{p})$.

This algorithm might not perform quite as expected, for two reasons. First, the heuristic analysis of the running time is not rigorous, and it is possible that the cycle of values, modulo p, could be much larger than $\sqrt{p}$. In this case, the algorithm performs correctly but much more slowly than desired. In practice, this issue seems to be moot. Second, the divisors of n produced by this algorithm might always be one of the trivial divisors 1 or n. For example, suppose that $n = pq$, where p and q are prime. It can happen that the values of t and u for p are identical with the values of t and u for q, and thus the factor p is always revealed in the same gcd operation that reveals the factor q. Since both factors are revealed at the same time, the trivial divisor $pq = n$ is revealed, which is useless. Again, this problem seems to be insignificant in practice. If necessary, we can restart the heuristic with a different recurrence of the form $x_{i+1} = (x_i^2 - c) \bmod n$. (We should avoid the values $c = 0$ and $c = 2$ for reasons we will not go into here, but other values are fine.)

Of course, this analysis is heuristic and not rigorous, since the recurrence is not really "random." Nonetheless, the procedure performs well in practice, and it seems to be as efficient as this heuristic analysis indicates. It is the method of choice for finding small prime factors of a large number. To factor a β-bit composite number n completely, we only need to find all prime factors less than $\lfloor n^{1/2} \rfloor$, and so we expect POLLARD-RHO to require at most $n^{1/4} = 2^{\beta/4}$ arithmetic operations and at most $n^{1/4}\beta^2 = 2^{\beta/4}\beta^2$ bit operations. POLLARD-RHO's ability to find a small factor p of n with an expected number $\Theta(\sqrt{p})$ of arithmetic operations is often its most appealing feature.

Exercises

31.9-1

Referring to the execution history shown in Figure 31.7(a), when does POLLARD-RHO print the factor 73 of 1387?

31.9-2

Suppose that we are given a function $f : \mathbb{Z}_n \to \mathbb{Z}_n$ and an initial value $x_0 \in \mathbb{Z}_n$. Define $x_i = f(x_{i-1})$ for $i = 1, 2, \ldots$. Let t and $u > 0$ be the smallest values such that $x_{t+i} = x_{t+u+i}$ for $i = 0, 1, \ldots$. In the terminology of Pollard's rho algorithm, t is the length of the tail and u is the length of the cycle of the rho. Give an efficient algorithm to determine t and u exactly, and analyze its running time.

31.9-3

How many steps would you expect POLLARD-RHO to require to discover a factor of the form p^e, where p is prime and $e > 1$?

31.9-4 $\star$

One disadvantage of POLLARD-RHO as written is that it requires one gcd computation for each step of the recurrence. Instead, we could batch the gcd computations by accumulating the product of several x_i values in a row and then using this product instead of x_i in the gcd computation. Describe carefully how you would implement this idea, why it works, and what batch size you would pick as the most effective when working on a β-bit number n.