For the key set 17, 138, 173, 294, 306, 472, 540, 551, and 618, Sprugnoli's algorithm finds the quotient reduction hash function (key + 25)/64, which yields the hash values 0, 2, 3, 4, 5, 7, 8, 9, and 10. The function is not minimal, since it distributes the 9 keys to a table of 11 positions. Sprugnoli's algorithm does, however, find the quotient reduction perfect hash function with the smallest table size.

An improvement to the algorithm yields a minimal perfect hash function of the form

```
h(key) = (key + s)/d           if key <= t
h(key) = (key + s + r)/d       if key > t
```

where the values s, d, t, and r are determined by the algorithm. However, the algorithm to discover such a minimal perfect hash function is $O(n^3)$ with a large constant of proportionality so that it is not practical for even very small key sets. A slight modification yields a more efficient algorithm that produces near-minimal perfect hashing function of this form for small key sets. In the foregoing example, such a function is

```
h(key) = (key - 7)/72          if key <= 306
h(key) = (key - 42)/72         if key > 306
```

which yields the hash values 0, 1, 2, 3, 4, 5, 6, 7, and 8 and happens to be minimal. A major advantage of quotient reduction hash functions and their variants is that they are order preserving.

Sprugnoli also presents another group of hashing functions, called ***remainder reduction perfect hash functions,*** which are of the form

```
h(key) = ((r + s * key) % x)/d
```

and an algorithm to produce values r, s, x, and d that yield such a perfect hash function for a given key set and a desired minimum load factor. If the minimum load factor is set to 1, a minimal perfect hash function results. However, the algorithm does not guarantee that a perfect remainder reduction hash function can be found in reasonable time for high load factors. Nevertheless, the algorithm can often be used to find minimal perfect hash functions for small key sets in reasonable time.

Unfortunately, Sprugnoli's algorithms are all at least $O(n^2)$ and are therefore only practical for small sets of keys (12 or fewer). Given a larger set of keys, k, a perfect hash function can be developed by a technique called ***segmentation***. This technique involves dividing k into a number of small sets, $k_0, k_2, \ldots, k_p$, and finding a perfect hash function h_i for each small set k_i. Assume a grouping function set such that key is in the set $k_{set(key)}$. If m_i is the maximum value of h_i on k_i and b_i is defined as $i + m_0 + m_1 + \cdots + m_{i-1}$, we can define the segmented hash function h as $h(key) = b_{set(key)} + h_{set(key)}(key)$. Of course, the function set that determines the grouping must be chosen with care to disperse the keys reasonably.

Jaeschke presents a method for generating minimal perfect hash functions using a technique called ***reciprocal hashing***. The reciprocal hash functions generated by

Jaeschke's algorithm are of the form

```
h(key) = (c/(d * key + e)) % tablesize
```

for some constants c, d, and e, and *tablesize* equal to the number of keys. Indeed, if the keys are all relatively prime integers, a constant c can be found that yields a minimal perfect hash function of the form

```
(c/key) % tablesize
```

by the following algorithm. Assume that the keys are initially in a sorted array $k(0)$ through $k(n-1)$ and that $f(c, key)$ is the function (c/key) % n.

```
c = ((n - 2) * k(0) * k(n - 1))/(k(n - 1) - k(0));
while (TRUE) {
   /*  check if c yields a perfect hash function    */
   bigi = -1;  /*  these will be set to the largest values    */
   bigj = -1;  /*  such that f(c, k(bigi)) = f(c, k(bigj))    */
   for (i = 0; i < n; i++)
      val(i) = f(c, k(i));
   for (i = n - 1; bigi < 0 && i >= 0; i--) {
      vi = val(i);
      j = i - 1;
      while (bigi < 0 && j >= 0)
         if (vi == val(j)) {
            bigi = i;
            bigj = j;
         }
         else
            j--;
   }  /* end for */
   if (bigi < 0)
      return;
   /*  increment c   */
   x = k(bigj) - (c % k(bigj));
   y = k(bigi) - (c % k(bigi));
   (x < y)  ?  c += x : c += y;
}  /* end while */
```

Applying this algorithm to the key set 3, 5, 11, 14 yields $c = 11$ and the minimal perfect hash function $(11/key)$ % 4. For the key set 3, 5, 11, 13, 14 the algorithm produces $c = 66$. In practice one would set an upper limit on the value of c to ensure that the algorithm does not go on indefinitely.

If the keys are not relatively prime, Jaeschke presents another algorithm to compute values d and e so that the values of $d * k(i) + e$ are relatively prime, so that the algorithm can be used on those values.

For low values of n, approximately 1.82^n values of c are examined by this algorithm, which is tolerable for $n <= 20$. For values of n up to 40, we can divide the keys into two sets $s1$ and $s2$ of size $n1$ and $n2$, where all keys in $s1$ are smaller than those of *s*2. Then we can find values *c*1, *d*1, *e*1 and *c*2, *d*2, *e*2 for each of the sets individually and use

```
h(key) = (c1/(d1 * key + e1)) % n1
```

for keys in *s*1 and

```
h(key) = n1 + (c2/(d2 * key + e2)) % n2
```

for keys in *s*2. For larger key sets, the segmentation technique of Sprugnoli can be used.

Chang presents an order-preserving minimal perfect hash function that depends on the existence of a ***prime number function,*** *p*(*key*), for the set of keys. Such a function always produces a prime number corresponding to a given key and has the additional property that if *key*1 is less than *key*2, *p*(*key*1) is less than *p*(*key*2). An example of such a prime number function is

```
p(x) = x² - x + 41      for    1 <= x <= 40
```

If such a prime number function has been found, Chang presents an efficient algorithm to produce a value *c* such that the function *h*(*key*) = *c* % *p*(*key*) is an order-preserving minimal hash function. However, prime number functions are difficult to find, and the value *c* is too large to be practically useful.

Cichelli presents a very simple method that often produces a minimal or near-minimal perfect hash function for a set of character strings. The hash function produced is of the form

```
h(key) = val(key[0]) + val(key[length(key) - 1]) + length(key)
```

where *val*(*c*) is an integer value associated with the character *c* and *key*[*i*] is the *i*th character of *key*. That is, add integer values associated with the first and last characters of the key to the length of the key. The integer values associated with particular characters are determined in two steps as follows.

The first step is to order the keys so that the sum of the occurrence frequencies of the first and last characters of the keys are in decreasing order. Thus if *e* occurs ten times as a last or first character, *g* occurs six times, *t* occurs nine times, and *o* occurs four times, the keys *gate*, *goat*, and *ego* have occurrence frequencies 16 (6 + 10), 15 (6 + 9), and 14 (10 + 4), respectively, and are therefore ordered properly.

Once the keys have been ordered, attempt to assign integer values. Each key is examined in turn. If the key's first or last character has not been assigned values, attempt to assign one or two values between 0 and some predetermined limit. If appropriate values can be assigned to produce a hash value that does not clash with the hash value of a previous key, tentatively assign those values. If not, or if both characters have been assigned values that result in a conflicting hash value, backtrack to modify tentative assignments made for a previous key. To find a minimal perfect

To illustrate Theorem 22.6, suppose that we have a string-matching problem with the pattern X being of length $n = 200$, which corresponds to 25 8-bit ASCII-encoded characters, and with the text string Y being of length 32000 corresponding to 4000 ASCII characters. Then $t = 31801$ and $M = nt^2 < 200(32000)^2 = 2^{11} \cdot 10^8 \approx 2^{11} \cdot 2^{26.6} < 2^{38}$ so that the base 2 computer representations of the primes p_i and corresponding residues will have fewer than 38 bits. If $k = 4$, then the probability of a false match will be less than about $2 \cdot 10^{-28}$.

The Karp and Rabin algorithm is a real-time algorithm requiring the same computation time for each text character. It also requires a constant amount of storage depending only upon n, k, and t. The algorithm can be proved to have a small probability of error. In addition, the method easily generalizes to apply to multidimensional problems. Other "fingerprinting" functions besides N_p may be appropriate.

We note that Karp and Rabin give other similar but different probabilistic pattern-searching algorithms in the reference cited earlier. For comparison purposes, information about some efficient nonprobabilistic searching algorithms may be found in (1) "Fast Pattern Matching in Strings," by D. E. Knuth, J. H. Morris, and V. R. Pratt [65]; (2) "A Fast String Searching Algorithm," by R. S. Boyer and J. S. Moore [12]; (3) by Jack Purdum "Pattern Matching Alternatives: Theory vs. Practice" [93]; and (4) "A Very Fast Substring Search Algorithm," by Daniel M. Sunday [111].

EXERCISES

1. Prove that if W and V are bit strings of length n, then $|N(W) - N(V)| < 2^n$.
2. In Theorem 22.2, the product of all primes less than or equal to w was shown to be greater than 2^w as long as $w \geq 29$. Determine which positive integers $w < 29$ have the property of the conclusion of this theorem.
3. In Theorem 22.3, two inequalities were shown to hold for every positive integer $w \geq 17$. By direct computation, determine which positive integers $w < 17$ also satisfy these inequalities.
4. For a pattern X of length $n = 160$ and a text of length $m = 20000$, use Theorem 22.6 to determine an upper bound on the probability of a false match for $k = 1, 2, 3$, and 4.
5. Assume that the pattern length n bits is given, that the number of primes k has been given, that M is such that all fingerprinting primes and their residues have fewer than 32 bits, and that the updating described in the discussion will be used. Determine the number of 32-bit computer words of information storage are needed to implement the Karp and Rabin algorithm.

22.2 APPLICATION: HASHING FUNCTIONS

In computer applications the situation frequently occurs in which there are n sets of information $I_1, I_2, \ldots, I_n$ where each set is "named" with a key, say $k_1, k_2, \ldots, k_n$, respectively. It is desirable to be able to find the information I_j quickly when the key k_j is given. The keys may be considered to be integers even though they may be finite sequences of alphabetic characters. Searching lists of keys may require too much time; moreover, if a search for information using the keys must be conducted many times, the computational work may be considerable. A compiler is a computer program that converts another computer program written in a high-level computer language such as FORTRAN or Pascal into a machine language program. During the process of "compiling," the compiler must produce tables and find items with the associated information in the tables many times. The important idea here is that the "lookup" table is created once but is used many times. Since there may be many such tables, one also needs to use the least storage feasible.

One solution to the "location" problem is to use a ***hash function*** $h : K \to \{0, 1, \ldots, m\}$ where $K = \{k_1, k_2, \ldots, k_n\}$ and where $m + 1 \geq n$. We then provide $m + 1$ computer storage locations where the n sets of information are kept. Thus, to find I_j, we calculate $h(k_j)$ and go to the location $h(k_j)$ where I_j is kept. It is generally difficult to find an ideal hash function, that is, one where $h(k_r) \neq h(k_s)$ when $k_r \neq k_s$ and where $m + 1 = n$. Hash functions that are often used map K into a set much larger than n elements and are not one-to-one. For h not one-to-one, provision must be made for what to do when there is collision, that is, when k_r and k_s are distinct keys but $h(k_r) = h(k_s)$. Donald E. Knuth in Volume 3 of *The Art of Computer Programming* [64] discusses various choices for hash functions and methods of dealing with collisions. G. Jaeschke in "Reciprocal Hashing: A Method for Generating Minimal Perfect Hashing Functions" [51] describes a particular type of minimum hashing function, that is, a hash function that is one-to-one and uses minimum storage (***minimum perfect hashing function***). Jaeschke's hashing function requires that one be able to map the keys $\{k_1, k_2, \ldots, k_n\}$, which may not be relatively prime to one another, to a set of integers $\{f(k_1), f(k_2), \ldots, f(k_n)\}$ that are pairwise relatively prime. He proved the following theorem.

THEOREM 22.7 Let $K = \{k_1, k_2, \ldots, k_n\}$ be a set of n distinct positive integers. There exist two integers D and E such that if $f(x)$ is defined by $f(x) = Dx + E$, then the members of the set of positive integers $\{f(k_1), f(k_2), \ldots, f(k_n)\}$ are pairwise relatively prime. Thus, the integers $Dk_1 + E$, $Dk_2 + E$, ..., and $Dk_n + E$ are pairwise relatively prime.

PROOF The proof is longer than we can present here (see Jaeschke's 1981 paper [51]); however, we will consider a special case later that will be adequate for our purposes. ▨

Jaeschke's main result is the next theorem, where $\lfloor w \rfloor$ is the greatest positive integer less than or equal to w.

THEOREM 22.8 If the set $K = \{k_1, k_2, \ldots, k_n\}$ has distinct positive integer keys, then there exist integers C, D, and E such that

$$h(x) = \left[\left[\left\lfloor \frac{C}{Dx + E} \right\rfloor\right]\right]_n \equiv \left\lfloor \frac{C}{Dx + E} \right\rfloor \pmod{n}$$

where $x \in K$, is a minimum perfect hashing function.

PROOF Let $k_1 < k_2 < \cdots < k_n$ and let the integers D and E be given by Theorem 22.7 so that $f(k_j) = Dk_j + E$ and $\gcd(f(k_i), f(k_j)) = 1$ for $i \neq j$. Since D and E may be chosen so that $f(k_j) > n$ for each j, it is possible to choose n integers $a_1, a_2, \ldots, a_n$ such that $a_i \not\equiv a_j \pmod{n}$ and such that

$$(i-1)\cdot(Dk_i + E) \le a_i < i \cdot (Dk_i + E)$$

Then, by the version of the Chinese remainder theorem applicable to nonrelatively prime moduli (Theorem 10.14), there is a number C such that

$$\begin{aligned} C &\equiv a_1 \pmod{n(Dk_1 + E)} \\ C &\equiv a_2 \pmod{n(Dk_2 + E)} \\ &\vdots \\ C &\equiv a_n \pmod{n(Dk_n + E)} \end{aligned}$$

Therefore, there are integers q_i such that $C = q_i[n(Dk_i + E)] + a_i$ for all i. Consequently,

$$\left\lfloor \frac{C}{Dk_i + E} \right\rfloor = q_i n + (i-1) \equiv i - 1 \pmod{n}$$

This result implies that the function h defined in the theorem has the property $h(k_i) \neq h(k_j)$ when $i \neq j$ so that h is one to one. Further, the range of the function h is $\{0, 1, \ldots, (n-1)\}$. ▩

Jaeschke gives algorithms for computing C, D, and E; however, the methods are exhaustive in nature. On the other hand, C. C. Chang and J. C. Shieh in "Pairwise Relatively Prime Generating Polynomials and Their Applications" [19] give a way to calculate a required polynomial of the form $f(x) = Dx + 1$ where $E = 1$. The proof of the next lemma is left to the reader.

LEMMA 22.9 If a and b are positive integers, $a > b$, and d is a multiple of $a - b$, then $d(a-b)$ and $da + 1$ are relatively prime.

THEOREM 22.10 Let $K = \{k_1, k_2, \ldots, k_n\}$ be a set of n distinct positive integers with $k_i < k_{i+1}$. If $\{t_1, t_2, \ldots, t_s\} = \{k_i - k_j : 1 \le j < i \le n\}$ is the set of $s = n(n-1)/2$ differences, then $D = w \cdot \operatorname{lcm}(t_1, t_2, \ldots, t_s)$, where w is any positive integer, has the property that $Dk_1 + 1, Dk_2 + 1, \ldots, Dk_n + 1$ are pairwise relatively prime.

PROOF Since $\gcd(a, b) = \gcd(a - b, a)$ when a and b are integers for which both $\gcd(a, b)$ and $\gcd(a - b, b)$ are defined, we obtain $\gcd(Dk_i + 1, Dk_j + 1) = \gcd(D(k_i - k_j), Dk_i + 1)$ for $i > j$, where D is as given in the theorem. By definition of D, D is a multiple of $(k_i - k_j)$; therefore, Lemma 22.9 gives $\gcd(Dk_i + 1, Dk_j + 1) = 1$ for $i > j$. ▩

We note that the proof of Theorem 22.10 requires only that D be a multiple of all of $t_1, t_2, \ldots, t_s$. Ordinarily, one would probably choose as small a D as possible ($w = 1$); however, the proof of Theorem 22.8 requires a D such that $f(k_i) = Dk_i + 1 > n$. Such a D can always be obtained by selecting w large enough.

EXAMPLE 22.11 Suppose that $K = \{3, 6, 7, 12\}$. Then

$$\begin{aligned}\{k_i - k_j : 1 \le j < i \le 4\} &= \{6-3, 7-3, 12-3, 7-6, 12-6, 12-7\} \\ &= \{3, 4, 9, 1, 6, 5\}\end{aligned}$$

so that $D = \text{lcm}(3, 4, 9, 1, 6, 5) = 180$. Thus, $f(x) = 180x + 1$ and

i	k_i	$f(k_i) = Dk_i + 1$
1	3	$180 \cdot 3 + 1 = 541$
2	6	$180 \cdot 6 + 1 = 1081$
3	7	$180 \cdot 7 + 1 = 1261$
4	12	$180 \cdot 12 + 1 = 2161$

By inspection, any two of 541, 1081, 1261, and 2161 are relatively prime. ■

In order to implement Jaeschke's reciprocal hashing we must be able to compute the integer C. Fortunately, C. C. Chang and J. C. Shieh in "A Fast Algorithm for Constructing Reciprocal Hashing Functions" [18] provide an algorithm for computing the integer C.

In the rest of this section we use both the greatest integer function $\lfloor w \rfloor$ described earlier and the ceiling function $\lceil w \rceil$, which is the least positive integer greater than or equal to w. Thus, $\lceil 5.23 \rceil = 6$ and $\lceil 8 \rceil = 8$.

THEOREM 22.12 If

(a) $m_1, m_2, \ldots, m_n$ are pairwise relatively prime integers,
(b) $m_i > n$ for all i, $1 \le i \le n$,
(c) $M = \prod_{j=1}^{n} m_j$,
(d) $M_i = n \prod_{j \ne i} m_j = \dfrac{nM}{m_i}$ for all i,
(e) b_i is such that $M_i b_i \equiv n \pmod{nm_i}$, and
(f) $N_i = \left\lceil \dfrac{(i-1)m_i}{n} \right\rceil$ for each i,

then

1.

$$\left\lfloor \frac{\sum_{j=1}^{n} M_j b_j N_j}{m_i} \right\rfloor \equiv (i-1) \pmod{n} \text{ for all } i$$

2.

$$C = \left[\left[\sum_{j=1}^{n} M_j b_j N_j \right]\right]_{nM}$$

is the smallest positive integer such that $\left\lfloor \frac{C}{m_i} \right\rfloor \equiv (i-1) \pmod{n}$ for all i.

PROOF Let $a_i = N_i \cdot n$. Then a_i is the smallest multiple of n such that $(i-1)\cdot m_i \leq a_i < i \cdot m_i$ for all i and $a_i \not\equiv a_j \pmod{n}$ when $i \neq j$. The sum $\sum_{j=1}^{n} M_j b_j N_j$ is a solution of the n congruences

$$\begin{aligned} x &\equiv a_1 \pmod{nm_1} \\ x &\equiv a_2 \pmod{nm_2} \\ &\vdots \\ x &\equiv a_n \pmod{nm_n} \end{aligned}$$

because if $j \neq i$, then M_j contains the factor nm_i; so $M_j b_j N_j \equiv 0 \pmod{nm_i}$ and

$$\begin{aligned} \sum_{j=1}^{n} M_j b_j N_j &\equiv M_i b_i N_i \pmod{nm_i} \\ &\equiv nN_i = a_i \pmod{nm_i} \end{aligned}$$

Therefore,

$$W_i = \frac{\sum\limits_{j=1}^{n} M_j b_j N_j}{m_i} = \frac{\sum\limits_{j \neq i}^{n} M_j b_j N_j}{m_i} + \frac{M_i b_i N_i}{m_i} = nJ_i + \frac{M_i b_i N_i}{m_i}$$

for an appropriate integer J_i. Since $M_i b_i \equiv n \pmod{nm_i}$, there exists an integer t_i such that $M_i b_i = t_i(nm_i) + n$ so that

$$W_i = nJ_i + t_i n N_i + \frac{nN_i}{m_i}$$

Thus, because $(i-1)\cdot m_i \leq a_i < i \cdot m_i$,

$$\lfloor W_i \rfloor \equiv \left\lfloor \frac{nN_i}{m_i} \right\rfloor \equiv \left\lfloor \frac{a_i}{m_i} \right\rfloor \equiv (i-1) \pmod{n}$$

which is part 1.

Let $C = \left[\left[\sum_{j=1}^{n} M_j b_j N_j\right]\right]_{nM}$. We need to show the congruence $\left\lfloor \frac{C}{m_i} \right\rfloor \equiv (i-1) \pmod{n}$ for $1 \leq i \leq n$. By the division algorithm there is an integer J such that $\sum\limits_{j=1}^{n} M_j b_j N_j = J(nM) + C$. For $1 \leq i \leq n$, again using the division algorithm, there are integers q_i and r_i with $0 \leq r_i < m_i$ such that

$$\sum_{j=1}^{n} M_j b_j N_j = q_i m_i + r_i$$

so that

$$\left\lfloor \frac{\sum_{j=1}^{n} M_j b_j N_j}{m_i} \right\rfloor = q_i$$

Therefore, by the result of part 1, there is an integer t_i such that

$$\begin{aligned} q_i &= (i-1) + t_i n \\ q_i m_i &= (i-1)m_i + t_i n m_i \\ q_i m_i + r_i &= (i-1)m_i + t_i n m_i + r_i \\ J(nM) + C &= (i-1)m_i + t_i n m_i + r_i \end{aligned}$$

so that

$$\frac{C}{m_i} = (i-1) + \left(t_i - \frac{JM}{m_i}\right)n + \frac{r_i}{m_i}$$

and

$$\begin{aligned} \left\lfloor \frac{C}{m_i} \right\rfloor &= (i-1) + \left(t_i - \frac{JM}{m_i}\right)n \\ \left\lfloor \frac{C}{m_i} \right\rfloor &\equiv (i-1) \pmod{n} \end{aligned}$$

It is left to the reader to show that if C' is any other positive integer such that $\left\lfloor \frac{C'}{m_i} \right\rfloor \equiv (i-1) \pmod{n}$ for all i, $1 \le i \le n$, then $C' > C$. ▩

Finally, we combine the last several results into the following theorem.

THEOREM 22.13 Let $K = \{k_1, k_2, \ldots, k_n\}$ be a set of n distinct positive integers; and let D and E be the integers, and $f(x) = Dx + E$ the function, described in Theorems 22.8 and 22.10. Also, assume that C is given by part 2 of Theorem 22.12 where $m_i = f(k_i)$. Then

$$h(x) = \left[\left[\left\lfloor \frac{C}{Dx+E} \right\rfloor \right]\right]_n$$

is a minimum perfect hashing function.

PROOF If $m_i = f(k_i)$, Theorem 22.12 implies that

$$h(k_i) = \left[\left[\left\lfloor \frac{C}{f(k_i)} \right\rfloor \right]\right] = (i-1) \text{ for } 1 \le i \le n$$

Thus, h is one-to-one and onto $\{0, 1, \ldots, (n-1)\}$. ▩

EXAMPLE 22.14

We now continue Example 22.11 where $K = \{k_1, k_2, k_3, k_4\} = \{3, 6, 7, 12\}$ and $f(x) = 180x + 1$. We found that $\{f(k_1), f(k_2), f(k_3), f(k_4)\} = \{541, 1081, 1261, 2161\}$. Let $m_i = f(k_i)$. The m_i's are pairwise relatively prime and $m_i > n = 4$. Using Theorem 22.12, we obtain the following table:

i	m_i	M_i	b_i	N_i	$M_i b_i N_i$	$h(k_i)$
1	541	11782990804	235	0	0	0
2	1081	5896945444	12	273	19318393274544	1
3	1261	5055192724	172	631	548650176721168	2
4	2161	2949837124	1303	1621	6230536829339212	3

For example, to find b_3 we solve $5055192724 \cdot b_3 \equiv 4 \pmod{5044}$, which is equivalent to solving $1263798181 \cdot b_3 \equiv 1 \pmod{1261}$ or to solving $22 \cdot b_3 \equiv 1 \pmod{1261}$. Therefore, we want b_3 and y such that $22b_3 + 1261y = 1$. Using the Euclidean Algorithm and back substitution, we obtain the result $22 \cdot 172 + (-3) \cdot 1261 = 1$. Thus

$$C = \left[\left[\sum_{j=1}^{n} M_j b_j N_j\right]\right]_{nM} = [[6798505399334924]]_{nM} = 3183904723300$$

Next we use C, f, and h to hash $k_3 = 7$.

$$h(k_3) = h(7) = \left[\left[\left\lfloor \frac{3183904723300}{1261} \right\rfloor\right]\right]_4 = [[2524904618]]_4 = 2 \qquad ■$$

Jaeschke's reciprocal hashing function generates large integers. The computational effort necessary to implement it should be compared to the resources required for using a nonminimal and nonperfect hashing function: (1) possible large memory requirements, (2) dealing with collisions, and (3) using search algorithms to find items in a table.

■ EXERCISES

1. If a and b are integers for which both $\gcd(a, b)$ and $\gcd(a - b, a)$ are defined, prove that $\gcd(a, b) = \gcd(a - b, a)$.
2. For each of these sets, find a polynomial $f(x) = Dx + 1$ that maps the set one-to-one and onto a set of pairwise relatively prime integers:
   (a) $\{12, 15, 18, 24\}$
   (b) $\{5, 15, 20, 25\}$
3. Let $a_1, a_2, \ldots, a_n$ be n distinct positive integers. Prove or disprove: $\gcd(a_1, a_2, \ldots, a_n) = 1$ if and only if $\gcd(a_i, a_j) = 1$ for all $i \neq j$.
4. In the proof of Theorem 22.12, show that $a_i = N_i n$ is the smallest multiple of n such that $(i - 1) \cdot m_i \leq a_i < i \cdot m_i$.
5. Verify the numbers in the table of Example 22.14.

[14] R. P. Brent and J. M. Pollard, "Factorization of the Eighth Fermat Number," *Math. of Comput.*, Volume 36, No. 154 (1981), 627-630.

[15] Richard A. Brualdi, *Introductory Combinatorics*, Prentice-Hall, Upper Saddle River NJ, 1999.

[16] F. Cajori, *A History of Mathematics*, 2nd ed., Macmillan, New York, 1961.

[17] R. D. Carmichael, "On Composite Numbers P Which Satisfy the Fermat Congruence $a^{P-1} \equiv 1 \bmod P$," *Amer. Math. Monthly*, Volume 19, No. 2 (1912), 22-27.

[18] C. C. Chang and J. C. Shieh, "A Fast Algorithm for Constructing Reciprocal Hashing Functions," *Proc. Internat. Symp. New Directions Comput.* (1985), 232-236.

[19] C. C. Chang and J. C. Shieh, "Pairwise Relatively Prime Generating Polynomials and Their Applications," *Proc. Internat. Workshop Discrete Algorithms Complexity*, November (1989), 137-139.

[20] Nan-xian Chen, "Modified Möbius Inversion Formula and Its Application to Physics," *Phys. Rev. Lett.*, Volume 64, No. 11 (1990), 1193-1195.

[21] L. W. Cohen and G. Ehrlich, *The Structure of the Real Number System*, Van Nostrand Reinhold, New York, 1963.

[22] H. Cohn, *Advanced Number Theory*, Dover, New York, 1980.

[23] D. Coppersmith, "Cryptography," *IBM J. Res. Develop.*, Volume 31, No. 2 (1987), 244-248.

[24] R. Crandall, J. Doenias, C. Norrie, and J. Young, "The Twenty-Second Fermat Number Is Composite," *Math. Comput.*, Volume 64 (1995), 863-868.

[25] T. Dantzig, *Number: The Language of Science*, 4th ed., Macmillan, New York, 1954.

[26] H. Davenport, *The Higher ABithmetic*, 6th ed., Cambridge University Press, New York, 1992.

[27] L. E. Dickson, *Studies in the Theory of Numbers*, Chelsea, New York, 1957.

[28] W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Trans. Inform. Theory*, Volume IT-22, No. 6 (1976), 644-654.

[29] H. W. Eves, *In Mathematical Circles, Quadrants I and II*, Prindle, Weber & Schmidt, Boston, 1969.

[30] William Feller, *An Introduction to Probability Theory and its Applications*, Wiley, New York, 1950.