JOURNAL OF ALGORITHMS **12**, 179–182 (1991)

NOTE

Two Algorithms for the Sieve Method

HERBERT S. WILF*

University of Pennsylvania, Philadelphia, Pennsylvania 19104-6395

Received May 2, 1990

We present two algorithms that implement the sieve method. The first one sieves *numbers*. It finds the *numbers* of objects that have exactly each number of properties. The second sieves *sets*. It finds the *sets* of objects that have exactly each number of properties. The algorithms are essentially identical, and both are direct descendants of Horner's method for solving polynomial equations, a method of numerically translating the roots, that dates back to 1819 (e.g., [1]). The algorithms differ in one respect at least. The first one is quite an efficient way to accomplish its task, while I cannot imagine a situation in which the second algorithm would be optimum from the point of view of efficiency.

1. THE "SIEVE NUMBERS" ALGORITHM

In more detail, we follow the scenario of [2]: we are given a set Ω of objects, and a collection $\mathcal{P}$ of properties that each object may or may not possess. For each subset $S \subseteq \mathcal{P}$ of properties, $N\ (\supseteq S)$ denotes the number of objects $\omega \in \Omega$ whose set of properties contains S, and $N_r = \sum_{|S|=r} N\ (\supseteq S)$, for each $r = 0, 1, 2, \ldots$. The numbers $\{N_r\}$ comprise the *input* to the sieve. The desired output quantities are the numbers $\{e_r\}$, where each e_r is the number of objects that have *exactly* r properties, for $r \geq 0$.

The "standard" method of computing the e's is by means of the sieve formulas

$$e_r = \sum_t (-1)^{t-r} \binom{t}{r} N_t \qquad (r = 0, 1, 2, \ldots).$$

*Research supported by the Office of Naval Research.

0196-6774/91 $3.00

We propose the following method:

ALGORITHM A. (Input $N_0, N_1, \ldots, N_q$. After running the algorithm, the input array of N's will have been transformed, in place, into the output array of e's.)

For $m = 0$ to $q - 1$
  For $i = q - 1$ downto m, do
    $N_i := N_i - N_{i+1}$ □

The total cost of the algorithm is clearly $\binom{q}{2}$ subtractions of integers, all binomial coefficient calculations having been avoided.

The reason that the algorithm works is the following: First, the relation between the power series generating functions $N(t) = \Sigma N_j t^j$ and $E(t) = \Sigma_j e_j t^j$ is just that $E(t) = N(t - 1)$ (e.g., [2, p. 101]). Hence if we have any method that will shift the origin of a Taylor expansion by a given amount, then that method can sieve. Horner's method is an old numerical method whereby one finds, say, the first significant digit of a root of a polynomial equation, and if that digit is, say 5, then we transform the equation into a new one in which all of the roots have been diminished by 5. In that way we are always dealing with an equation whose next root is near the origin, which gives good efficiency in root finding, or anyway, it did in 1819.

The precise algorithm (iterated synthetic division) by which a Taylor expansion is transformed into another one that represents the same function about a new origin is "Algorithm Taylor" on page 173 of [3]. Algorithm A above is just the special case in which we translate the origin by one unit. A self-contained combinatorial proof of Algorithms A and B appears below.

As an example of the algorithm, consider the set Ω to be the six permutations of three letters. For each $i = 1, 2, 3$, say that a permutation σ has property i if $\sigma(i) = i$. Then

$$(N_0, N_1, N_2, N_3) = (6, 6, 3, 1).$$

In Table 1, the columns show the status of the N array at the start, and after each pass (= value of m) of Algorithm A. The last column shows the

TABLE 1

	Start	1st pass	2nd pass	3rd pass
N_0	6	2	2	2
N_1	6	4	3	3
N_2	3	2	1	0
N_3	1	1	1	1

number of objects with exactly j properties, for each j. Notice that, for instance, e_0 is available after one pass, and in general, e_r is available after $r + 1$ passes of the algorithm.

2. The "Sieve-Sets" Algorithm

The simple form of Algorithm A, in which the only arithmetic on display is a single subtraction of integers, suggests that it might have a set-theoretic counterpart, in which the subtraction of two integers is replaced by the subtraction of two multisets. Suppose we want not just the *numbers* of objects that have exactly each number of properties, but also we want to see the *sets* of objects that have exactly each number of properties. Then instead of beginning with the overcounted numbers N_i we begin with overpopulated multisets of objects. The single "$-$" sign in Algorithm A is interpreted as a subtraction of multisets. The algorithm terminates with the desired sets.

More precisely, for each $i \geq 0$ we construct a multiset A_i as follows. A_i is initially empty. For each set S of cardinality i, adjoint to the multiset A_i the collection of all objects $\omega \in \Omega$ whose set of properties contains S. Clearly $|A_i| = N_i$, for all i. These multisets are the input to Algorithm B, below. Its output is a collection of *sets* $E_0, E_1, \dots$ such that each E_j is the set of $\omega \in \Omega$ that have exactly j properties. The algorithm is formally identical to Algorithm A above.

Algorithm B. (Input are the multisets $A_0, A_1, \dots, A_q$. They are transformed, in place, into the output sets $E_0, E_1, \dots$.)

For $m = 0$ to $q - 1$
  For $i = q - 1$ downto m, do
    $A_i := A_i - A_{i+1}$ □

At termination, the multisets A_i will have been sieved into the sets E_i. In the example above, of the permutations of three letters, the following table shows the status of the collection of multisets of permutations at the start of Algorithm B, and after each pass (= value of m). Permutations are displayed by their values.

We observe that at each corresponding stage, an entry in Table 1 shows the cardinality of the corresponding multiset in Table 2, and therefore a proof of correctness of Algorithm B is *a fortiori* a proof of correctness of Algorithm A. We now prove that Algorithm B works as claimed.

Proofs of Correctness. Fix an object $\omega \in \Omega$, and suppose that ω has exactly p properties. The multiplicity with which ω appears in a multiset

TABLE 2

	Start	1st pass	2nd pass	3rd pass
	123	231	231	231
	132	312	312	312
A_0	213			
	231			
	312			
	321			
	123	132	132	132
	132	321	321	321
A_1	123	123	213	213
	321	213		
	123			
	213			
	123	123	123	∅
A_2	123	123		
	123			
A_3	123	123	123	123

A_i is $\binom{p}{i}$, for $i \geq 0$, at the start of Algorithm B. After the rth pass of the algorithm, ω appears in multiset A_i with multiplicity

$$\begin{cases} \binom{p-r}{i-r}, & \text{if } r \leq p; \\ 1, & \text{if } r > p \text{ and } i = p; \\ 0, & \text{if } r > p \text{ and } i \neq p, \end{cases}$$

as one readily checks by induction on r.

Hence after every pass $r \geq p$, object ω belongs to just one multiset, namely A_p, and it occurs there with multiplicity 1. Thus, if the number of passes exceeds the largest number of properties that any object has, then every object lives uniquely in the multiset whose subscript is the number of properties that that object has, and all of the "multisets" are then, in fact, sets. □

References

1. L. E. Dickson, "First Course in the Theory of Equations," Wiley, New York, 1922.
2. H. S. Wilf, "generatingfunctionology," Academic Press, Boston, 1990.
3. A. Nijenhuis and H. S. Wilf, "Combinatorial Algorithms" (2nd ed.), Academic Press, New York, 1978.