8.1 Computing a Binomial Coefficient

Computing a binomial coefficient is a standard example of applying dynamic programming to a nonoptimization problem. You may recall from your studies of elementary combinatorics that the ***binomial coefficient***, denoted $C(n, k)$ or $\binom{n}{k}$, is the number of combinations (subsets) of k elements from an n-element

set ($0 \le k \le n$). The name "binomial coefficients" comes from the participation of these numbers in the binomial formula:

$$(a + b)^n = C(n, 0)a^n + \cdots + C(n, k)a^{n-k}b^k + \cdots + C(n, n)b^n.$$

Of the numerous properties of binomial coefficients, we concentrate on two:

$$C(n, k) = C(n - 1, k - 1) + C(n - 1, k) \quad \text{for } n > k > 0 \tag{8.3}$$

and

$$C(n, 0) = C(n, n) = 1. \tag{8.4}$$

The nature of recurrence (8.3), which expresses the problem of computing $C(n, k)$ in terms of the smaller and overlapping problems of computing $C(n - 1, k - 1)$ and $C(n - 1, k)$, lends itself to solving by the dynamic programming technique. To do this, we record the values of the binomial coefficients in a table of $n + 1$ rows and $k + 1$ columns, numbered from 0 to n and from 0 to k, respectively (Figure 8.1).

To compute $C(n, k)$, we fill the table in Figure 8.1 row by row, starting with row 0 and ending with row n. Each row i ($0 \le i \le n$) is filled left to right, starting with 1 because $C(n, 0) = 1$. Rows 0 through k also end with 1 on the table's main diagonal: $C(i, i) = 1$ for $0 \le i \le k$. We compute the other entries by formula (8.3), adding the contents of the cells in the preceding row and the previous column and in the preceding row and the same column. (If you recognize Pascal's triangle—a fascinating mathematical structure usually studied in conjunction with the notion of a combination—you are right: this is exactly what it is.) The following pseudocode implements this algorithm.

	0	1	2	...	$k-1$	k
0	1					
1	1	1				
2	1	2	1			
⋮						
k	1					1
⋮						
$n-1$	1				$C(n-1, k-1)$	$C(n-1, k)$
n	1					$C(n, k)$

FIGURE 8.1 Table for computing the binomial coefficient $C(n, k)$ by the dynamic programming algorithm

ALGORITHM *Binomial*(n, k)

```
//Computes C(n, k) by the dynamic programming algorithm
//Input: A pair of nonnegative integers n ≥ k ≥ 0
//Output: The value of C(n, k)
for i ← 0 to n do
    for j ← 0 to min(i, k) do
        if j = 0 or j = i
            C[i, j] ← 1
        else C[i, j] ← C[i − 1, j − 1] + C[i − 1, j]
return C[n, k]
```

What is the time efficiency of this algorithm? Clearly, the algorithm's basic operation is addition, so let $A(n, k)$ be the total number of additions made by this algorithm in computing $C(n, k)$. Note that computing each entry by formula (8.3) requires just one addition. Also note that because the first $k + 1$ rows of the table form a triangle while the remaining $n - k$ rows form a rectangle, we have to split the sum expressing $A(n, k)$ into two parts:

$$A(n, k) = \sum_{i=1}^{k} \sum_{j=1}^{i-1} 1 + \sum_{i=k+1}^{n} \sum_{j=1}^{k} 1 = \sum_{i=1}^{k} (i - 1) + \sum_{i=k+1}^{n} k$$
$$= \frac{(k - 1)k}{2} + k(n - k) \in \Theta(nk).$$

You are asked to ascertain whether this is an efficient algorithm by comparing it with the running times of a few other algorithms for this problem in the exercises. Another problem in the exercises is to analyze whether or not the extra space used by the dynamic programming algorithm is actually necessary.

Exercises 8.1

1. **a.** What does dynamic programming have in common with divide-and-conquer?
   b. What is a principal difference between the two techniques?
2. **a.** Compute $C(6, 3)$ by applying the dynamic programming algorithm.
   b. Is it also possible to compute $C(n, k)$ by filling the algorithm's dynamic programming table column by column rather than row by row?
3. Prove the following assertion made in the text while investigating the time efficiency of the dynamic programming algorithm for computing $C(n, k)$:

$$\frac{(k - 1)k}{2} + k(n - k) \in \Theta(nk).$$