2.5 Example:Computing the *n*th Fibonacci Number

In this section, we consider the *Fibonacci numbers*, a famous sequence

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots \tag{2.5}$$

that can be defined by the simple recurrence

$$F(n) = F(n-1) + F(n-2) \qquad \text{for } n > 1 \tag{2.6}$$

and two initial conditions

$$F(0) = 0, \qquad F(1) = 1. \tag{2.7}$$

To start, let us get an explicit formula for $F(n)$. If we try to apply the method of backward substitutions to solve recurrence (2.6), we will fail to get an easily discernible pattern. Instead, we can take advantage of a theorem that describes solutions to a *homogeneous second-order linear recurrence with constant coefficients*

$$ax(n) + bx(n-1) + cx(n-2) = 0, \qquad (2.8)$$

where a, b, and c are some fixed real numbers ($a \neq 0$) called the coefficients of the recurrence and $x(n)$ is the generic (通用的) term unknown sequence to be found.

Applying this theorem to our recurrence with the initial conditions given—see Appendix B—we obtain the formula

$$F(n) = \frac{1}{\sqrt{5}}(\phi^n - \hat{\phi}^n), \qquad \textbf{(2.9)}$$

where $\phi = (1+\sqrt{5})/2$ **≈ 1.61803 and** $\hat{\phi} = -1/\phi$ **≈ −0.61803.**[6] **It is hard to believe that formula (2.9), which includes arbitrary integer powers of irrational numbers, yields nothing else but all the elements of Fibonacci sequence (2.5), but it does!**

One of the benefits of formula (2.9) is that it immediately implies that $F(n)$ grows exponentially (remember Fibonacci's rabbits?), i.e., $F(n) \in \Theta(\varphi^n)$. This follows from the observation that $\hat{\phi}$ is a fraction between −1 and 0, and hence $\hat{\phi}^n$ gets infinitely small as n goes to infinity. In fact, one can prove that the impact of the second term $\frac{1}{\sqrt{5}}\hat{\phi}^n$ on the value of $F(n)$ can be obtained by rounding off the value of the first term to the nearest integer. In other words, for every nonnegative integer n,

$$F(n) = \frac{1}{\sqrt{5}}\phi^n \quad \text{rounded to the nearest integer.} \qquad \textbf{(2.10)}$$

```
ALGORITHM    $F(n)$
    //Computes the $n$th Fibonacci number recursively by using its
      definition
    //Input: A nonnegative integer $n$
    //Output: The $n$th Fibonacci number
    if $n \le 1$ return $n$
    else return $F(n - 1) + F(n - 2)$
```

Before embarking on its formal analysis, can you tell whether this is an efficient algorithm?Well, we need to do a formal analysis anyway. The algorithm's basic operation is clearly addition, so let $A(n)$ be the number of additions performed by the algorithm in computing $F(n)$. Then the numbers of additions needed for computing $F(n-1)$ and $F(n-2)$ are $A(n-1)$ and $A(n-2)$, respectively, and the algorithm needs one more addition to compute their sum. Thus, we get the following recurrence for $A(n)$:

$$A(n) = A(n-1) + A(n-2) + 1 \quad \text{for } n > 1, \qquad (2.11)$$
$$A(0) = 0, \qquad A(1) = 0.$$

The recurrence $A(n) - A(n-1) - A(n-2) = 1$ is quite similar to recurrence $F(n) - F(n-1) - F(n-2) = 0$, but its right-hand side is not equal to zero. Such recurrences are called ***inhomogeneous.*** There are general techniques for solving inhomogeneous recurrences (see Appendix B or any textbook on discrete mathematics), but for this particular recurrence, a special trick leads to a faster solution. We can reduce our inhomogeneous recurrence to a homogeneous one by rewriting it as

$$[A(n) + 1] - [A(n-1) + 1] - [A(n-2) + 1] = 0$$

and substituting $B(n) = A(n) + 1$:

$$B(n) - B(n-1) - B(n-2) = 0,$$
$$B(0) = 1, \qquad B(1) = 1.$$

This homogeneous recurrence can be solved exactly in the same manner as recurrence (2.6) was solved to find an explicit formula for $F(n)$. But it can actually be avoided by noting that $B(n)$ is, in fact, the same recurrence as $F(n)$ except that it starts with two 1's and thus runs one step ahead of $F(n)$. So $B(n) = F(n + 1)$, and

$$A(n) = B(n) - 1 = F(n+1) - 1 = \frac{1}{\sqrt{5}}\left(\phi^{n+1} - \hat{\phi}^{n+1}\right) - 1.$$

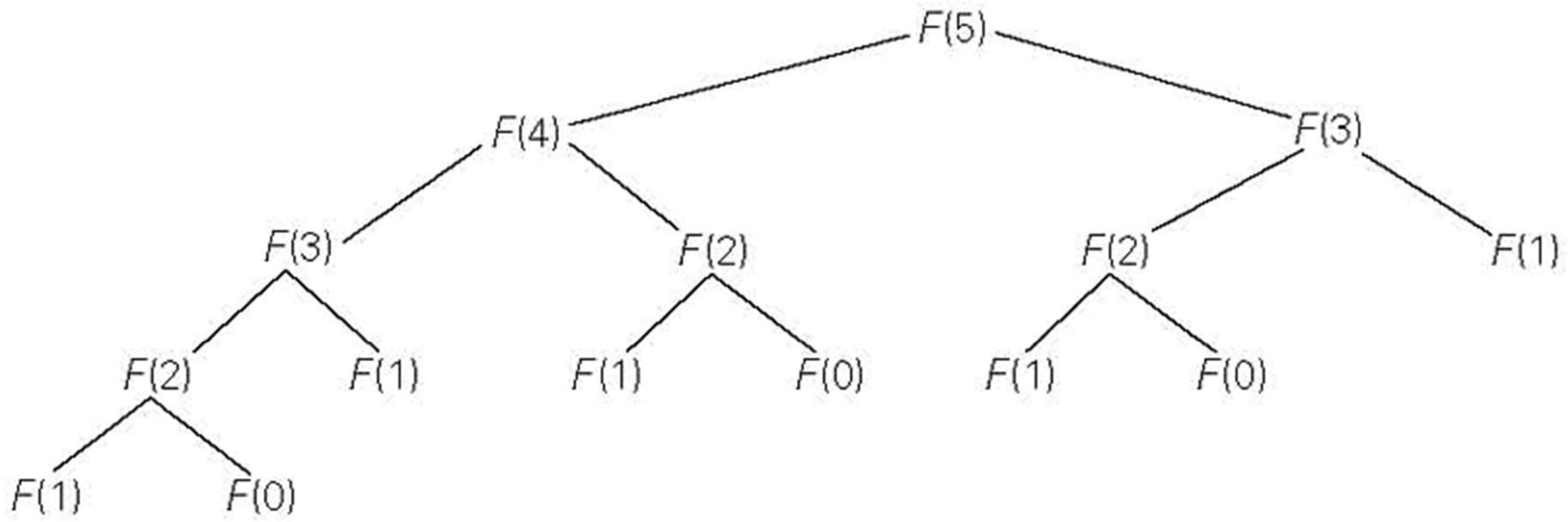

FIGURE 2.6 Tree of recursive calls for computing the 5th Fibonacci number by the definition-based algorithm.

```
ALGORITHM    $Fib(n)$
    //Computes the $n$th Fibonacci number iteratively by using its
     definition
    //Input: A nonnegative integer $n$
    //Output: The $n$th Fibonacci number
    $F[0] \leftarrow 0$; $F[1] \leftarrow 1$
    for $i \leftarrow 2$ to $n$ do
            $F[i] \leftarrow F[i-1] + F[i-2]$
    return $F[n]$
```

Finally, there exists a $\Theta(\log n)$ algorithm for computing the nth Fibonacci number that manipulates only integers. It is based on the equality

$$\begin{bmatrix} F(n-1) & F(n) \\ F(n) & F(n+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n \text{ for } n \geq 1$$

and an efficient way of computing matrix powers.