Sections 3.2: Recurrence Relations

February 17, 2025

Abstract

Recurrence relations are defined recursively, and solutions can sometimes be given in “closed-form” (that is, without recourse to the recursive definition). We will solve one type of linear recurrence relation to give a general closed-form solution, the solution being verified by induction.

We’ll be getting some practice with summation notation in this section. (Have you seen it before?)

$!$: $0! = 1$

$n! = n \cdot (n-1)!$

$s(n) = f_1(n)\, s(n-1)$

1 Solving Recurrence Relations

Vocabulary:

- **linear recurrence relation:** $S(n)$ depends linearly on previous $S(r)$, $r < n$:

$$S(n) = f_1(n)S(n-1) + \cdots + f_k(n)S(n-k) + g(n)$$

  That means no powers on $S(r)$, or any other functions operating on $S(r)$. The relation is called **homogeneous** if $g(n) = 0$. (Both Fibonacci numbers and factorials are defined by homogeneous linear recurrence relations.)

F: $F(1) = 1$

$F(2) = 1$

$F(n) = F(n-1) + F(n-2)$

Both are homogeneous: no $g(n)$

- **first-order:** $S(n)$ depends only on $S(n-1)$, and not previous terms. (Factorials are first-order, while Fibonaccis are second-order, depending on the two previous terms.)

2nd order - requires two preceding values.

- **constant coefficient:** In the linear recurrence relation, when the coefficients of previous terms are constants. (Fibonaccis are constant coefficient; factorials are not.)

- **closed-form solution**: $S(n)$ is given by a formula which is simply a function of n, rather than a recursive definition of itself. (Fibonacci numbers have a closed-form solution; I would say factorials, not so much....)

The author suggests an “expand, guess, verify” method for solving recurrence relations.

Example: The story of T

(a) Practice 1, p. 159 (from the previous section):

$$T(1) = 1$$
$$T(n) = T(n-1) + 3, \text{ for } n \geq 2$$

closed form solution.

(b) Practice 9, p. 168: Here is the recurrence relation for Example 11, p. 130, in lisp:

```
(defun Tee(n)
  (if (integerp n)
      (cond
       ((>= n 2)
        (+ (Tee (- n 1)) 3)
        )
       ((= n 1)
        1
        )
       (t (error "Tilt! Only positive ints allowed in function tee..."))
       )
    (error "Tilt! Only positive integers allowed in function tee...")
    )
  )
> (tee 2)
4
> (mapcar #'tee (iseq 1 10))
(1 4 7 10 13 16 19 22 25 28)
```

$P(n)$: $\boxed{T(n) = 3(n-1)+1}$

$P(1)$: $T(1) = 3(1-1)+1 = 1$ ✓

Assume $P(k)$, & consider $P(k+1)$: the assertion that $T(k+1) = 3((k+1)-1)+1$. Consider the LHS:

$T(k+1) = T(k) + 3$ (from RR)

$= 3(k-1)+1 + 3$ (by assumption)

$= 3[(k-1)+1]+1$

$= 3[(k+1)-1]+1$ ✓

(the RHS).

(c) Practice 11, p. 181: Find a closed-form solution for the recurrence relation for sequence T of part (a).

Example: general linear first-order recurrence relations with constant coefficients.

$$S(1) = a$$
$$S(n) = cS(n-1) + g(n), \ n \in \{2, 3, 4, \ldots\}$$

"Expand, guess, verify" (then prove by induction!):

$$S(n) = c^{n-1}S(1) + \sum_{i=2}^{n} c^{n-i}g(i) \qquad (1)$$

Now check that this formula works for $T(n)$ from above.

2 Counting Using Recurrence Relations

Algorithm *BinarySearch* (which is discussed in the previous section) is recursive: it calls itself. Starting from a list of length n it makes one comparison and then calls itself with a list of half its initial length. Hence the number of comparisons for the list of length n, $C(n)$, would be (in the worst case)

$n = 5$

$1 \quad 3 \quad 7 \quad 9 \quad 11$

$$C(n) = C(floor(n/2)) + 1 :$$

that is, you'd need to check the middle element, then do a binary search of the sorted list to the left or right, of half the length (or so) of the original list. For a list of length 1, we have our base case: $C(1) = 1$.

$8 : 7$

That floor function in the inductive step is a pain, but is necessary since n may be odd.

$1 \quad 3 \quad 7 \quad 9$

Forgetting the floor for the moment, use the "expand, guess, and verify" approach: in the worst-case scenario, the algorithm will find the element (or not) on its last check (when it's down to a list of length 1).

$$C(n) = C(n/2) + 1 = (C(n/4) + 1) + 1 = ((C(n/8) + 1) + 1) + 1 = ...$$

$8 : 3$

Obviously this is only going to work easily (in the sense that $C(n/8)$, etc., make sense) if n is a power of 2. Assume therefore that $n = 2^m$, for integer m. This allows us to throw away the floor function, and makes all quotients reasonable.

Before we begin, can you guess how many comparisons we make in the worst case, for $C(n)$ when $n = 2^m$?

Let's consider a change of variable. First of all, we replace n by 2^m:

$$C(2^m) = C(2^m/2) + 1 = \underline{C(2^{m-1})} + 1.$$

Then we define $T(m) = C(2^m)$ (think of T as a composition of functions, $C(x)$ and 2^x); hence

$T(m-1) = C(2^{m-1})$

$$T(m) = T(m-1) + 1$$

Note that $T(0) = C(1) = 1$. We can solve easily to get a closed-form solution:

$$T(m) = m + 1$$

$n = 2^m \iff m = \log_2 n$

when $n = 1$

$m = 0$

Let's now re-express that in terms of C and n. Since $n = 2^m$, we can equally well write $m = log_2(n)$. Hence, $C(n) = C(2^m) = T(m) = m+1 = log_2(n)+1$. This compares quite favorably with the worst-case estimate from *SequentialSearch*, which would be n (linear in n).

$C(n) = C(2^m)$

$= T(m)$

$= m + 1$

(For those of you who've forgotten, the log function grows much more slowly than a linear function does.)

$\boxed{C(n) = \log_2 n + 1}$

Let's look at the general recurrence relation of the "divide and conquer" variety: given

$$\begin{aligned} S(1) &= a \\ S(n) &= cS(n/2) + g(n) \end{aligned} \qquad (2)$$

Assume $n = 2^m$ for some integer m. Then

$$\cancel{S(2^0) =} aS(2^m) = cS(2^{m-1}) + g(2^m)$$

$$S(2^m) = cS(2^{m-1}) + g(2^m)$$

$$T(m) = cT(m-1) + g(2^m)$$

Now let's perform a change of variables: let $T(m) = S(2^m)$, so that

$$\begin{aligned} &T(0) = a \\ &T(m) = cT(m-1) + g(2^m) \end{aligned}$$

Using formula (1) (formula 8, p. 183), we get

$$T(m) = c^{m-1}\boxed{T(1)} + \sum_{i=2}^{m} c^{m-i} g(2^i)$$

$$T(1) = cT(0) + g(2^1)$$

Then reindexing, since we start with 0 rather than 1, we get

$$T(m) = c^{m-1}\left(cT(0) + g(2^m)\right) + \sum_{i=2}^{m} c^{m-i} g(2^i)$$

$$T(m) = c^m T(0) + \sum_{i=1}^{m} c^{m-i} g(2^i)$$

Finally, substituting back in S and n, we get

$$\boxed{S(n) = c^{\log_2 n} a + \sum_{i=1}^{\log_2 n} c^{\log_2 n - i} g(2^i)}$$

$$= c^m T(0) + c^{m-1} g(2^1) + \sum_{i=2}^{m} c^{m-i} g(2^i)$$

$$= c^m T(0) + \sum_{i=1}^{m} c^{m-i} g(2^i)$$

Whew! This is the general solution for the divide-and-conquer algorithm of type (2).

Example: Exercise #46, p. 202

$$\begin{aligned} &S(1) = 3 \\ &S(n) = S(\tfrac{n}{2}) + n \quad \text{for } n \geq 2, n = 2^m \end{aligned}$$