

Sections 3.2: Recurrence Relations

February 10, 2020

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?

1 Solving Recurrence Relations

Vocabulary:

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

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

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 and factorial are examples of homogeneous linear recurrence relations.)

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

$$G(n) = n G(n-1)$$

- **first-order:** $S(n)$ depends only on $S(n-1)$, and not previous terms. (Factorial is first-order, while Fibonacci is second-order, depending on the two previous terms.)
- **constant coefficient:** In the linear recurrence relation, when the coefficients of previous terms are constants. (Fibonacci is constant coefficient; factorial is 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. (Both Fibonacci and factorial have closed-form solutions.)

Given:

$$S(1) = a$$

$$S(n) = c \cdot S(n-1) + g(n)$$

$$S(2) = c \cdot S(1) + g(2)$$

$$= c \cdot a + g(2)$$

$$S(3) = c \cdot S(2) + g(3)$$

$$= c(c \cdot a + g(2)) + g(3)$$

$$= c^2 a + c g(2) + g(3)$$

$$S(4) = c S(3) + g(4)$$

$$= c(c^2 a + c g(2) + g(3)) + g(4)$$

$$= c^3 a + c^2 g(2) + c g(3) + g(4)$$

$$S(n) = c^{n-1} a + \underbrace{c^{n-2} g(2)} + \underbrace{c^{n-3} g(3)} + \dots + \underbrace{c g(n-1)} + \underbrace{g(n)}$$

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

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

$$= 1 \cdot a = S(1)$$

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

$$= c \cdot a + c^{2-2} g(2)$$

$$= c a + g(2) \checkmark$$

Assume $P(k)$:

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

Consider $P(k+1)$; $k+1 \in \mathbb{N}$:

$$S(k+1) = c S(k) + g(k+1)$$

$$= c \left(c^{k-1} a + \sum_{i=2}^k c^{k-i} g(i) \right) + g(k+1)$$

$$= c^{(k+1)-1} a + \sum_{i=2}^k c^{(k+1)-i} g(i) + g(k+1)$$

$$= c^{(k+1)-1} a + \sum_{i=2}^{k+1} c^{(k+1)-i} g(i)$$

$\therefore P(k+1)$

And by the 1st principle of induction, $P(n) \forall n \in \mathbb{N}$.

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):

$$T(1) = 1$$

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

First order
constant coefficient
non-homogeneous.
 $g(n) = 3$

(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)
```

recurrence relation

$$T(1) = 1$$

$$T(2) = T(1) + 3 = 1 + 3$$

$$T(3) = T(2) + 3 = (1 + 3) + 3 = 1 + 3 + 3$$

 $T(4) = T(3) + 3 = 1 + 3 + 3 + 3$

$T(n) = 1 + (n-1) \cdot 3$ Guess

By induction: $T(2)$ base case ✓

Assume $T(k) = 1 + (k-1) \cdot 3 = P(k)$

Consider $P(k+1)$, LHS:

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

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

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

$$P(k+1). \checkmark$$

∴ Therefore $P(n)$
 $\forall n \in \mathbb{N}$.

(c) Practice 11, p. 181

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

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

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

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

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

$$T(1) = 1$$

$$T(n) = T(n-1) + 3$$

$a=1$
 $c=1$
 $g(n)=3$

$$T(n) = \underbrace{1}_{1} + \sum_{i=2}^n \underbrace{3}_{n-1 \text{ times}} = 1 + (n-1) \cdot 3$$

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)

$$C(n) = C(\text{floor}(n/2)) + 1$$

(+ one comparison, don't find it);

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: $C(1) = 1$.

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

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).

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

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 $C(n)$?

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

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

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

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

$T(0) = 1$
 $T(1) = 2$
 $T(2) = 3 \dots$

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

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

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

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).

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

$$C(2^m) = m + 1$$

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

$$C(n) = \log_2 n + 1$$

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

$$S(1) = a$$

$$S(n) = cS(n/2) + g(n)$$

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

$$S(2^0) = a$$

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

Now we perform the change of variables: let $T(m) = S(2^m)$, so that

$$T(0) = a$$

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

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

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

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

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

Finally, substituting back in S and n , we get

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

Whew!

Example: Exercise #46, p. 202 (using variable S rather than the T that they used)

$$S(1) = 3$$

$$S(n) = S(n/2) + n \quad \text{for } n \geq 2, n = 2^m$$

$$S(2) = S(1) + 2 = 5$$

$$S(4) = S(2) + 4 = 9$$

$$S(n) = 1 \cdot 3 + \sum_{i=1}^{\log_2 n} 1 \cdot 2^i$$

$$= 3 + \sum_{i=1}^{\log_2 n} 2^i$$

$$= 3 + 2(n-1)$$

$$S(n) = 1 + 2n$$

$$S(2) = 5 \quad S(4) = 9 \quad \checkmark$$

$$T(n) = c^{n-1} [cT(0) + g(2^1)] + \sum_{i=2}^n c^{n-i} g(2^i)$$

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

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

should have used something other than S!
cool trick

$$S = 1 + r + r^2 + \dots + r^k$$

$$rS = r + r^2 + \dots + r^k + r^{k+1}$$

$$S - rS = 1 - r^{k+1}$$

$$(1-r)S = 1 - r^{k+1}$$

$$S = \frac{1 - r^{k+1}}{1 - r}$$

$$2^1 + 2^2 + \dots + 2^{\log_2 n}$$

$$= 2(1 + 2 + \dots + 2^{\log_2 n - 1})$$

$$= 2 \frac{1 - 2^{\log_2 n}}{1 - 2} = 2(2^{\log_2 n} - 1) = n$$