34 Counting Using Recurrence Relations

It is not always convenient to use the methods of earlier chapters to solve counting problems. Another technique for finding the solution to a counting problem is **recursive counting**. The method will be illustrated with several examples.

34.1 Recursive counting method

Example 34.1. *Recall that a bit string is a list of* 0*'s and* 1*'s, and the length of a bit string is the total number of* 0*'s and* 1*'s in the string. For example,* 10111 *is a bit string of length five, and* 000100 *is a bit string of length six. The problem of counting the number of bit strings of length n is duck soup. There are two choices for each bit, and so, applying the product rule, there are 2^n such strings. However, consider the problem of counting the number of bit strings of length n with no adjacent* 0*'s.*

Let's use a_n to denote the number of bit strings of length n with no adjacent 0*'s. Here are a few sample cases for small values of n.*

n=0: *Just one good bit string of length zero, and that is λ, the empty bit string. So $a_0 = 1$.*

n=1: *There are two good bit strings of length one. Namely* 0 *and* 1*. So $a_1 = 2$.*

n=2: *There are three good bit strings of length two. Namely,* 01, 10 *and* 11*. (Of course,* 00 *is a bad bit string.) That means $a_2 = 3$.*

n=3: *Things start to get confusing now. But here is the list of good bit strings of length three:* 010, 011, 101, 110*, and* 111*. So $a_3 = 5$.*

n=4: *A little scratch work produces the good bit strings* 0101, 0111, 1011, 1101, 1111, 0110, 1010, *and* 1110, *for a total of eight. That means* $a_4 = 8$.

We can do a few more, but it is hard to see a formula for a_n *like the* 2^n *formula that gives the total number of all bit strings of length* n*. Even though a formula for* a_n *is difficult to spot, there is a pattern to the list of values for* a_n *which looks like the Fibonacci sequence pattern. In fact, the list so far looks like* 1, 2, 3, 5, 8, *and if a few more are worked out by brute force, it turns out the list continues* 13, 21, 34*. So, it certainly seems that the solution to the counting problem can be expressed recursively as* $a_0 = 1$, $a_1 = 2$, *and for* $n \geq 2$, $a_n = a_{n-1} + a_{n-2}$*. If this guess is really correct, then we can quickly compute the number of good bit strings of length* n*. We just calculate* a_0, a_1, a_2*, etc., until we reach the* a_n *we are interested in.*

Such a recursive solution to counting problems is certainly less satisfactory than a simple formula, but some counting problems are so messy that a simple formula might not be possible, and the recursive solution is better than nothing in such a case.

There is one problem with the recursive solution offered in example 34.1. We said that *it seems that* the solution to the counting problem can be expressed recursively as $a_0 = 1$, $a_1 = 2$, and for $n \geq 2$, $a_n = a_{n-1} + a_{n-2}$. That *it seems that* is not an acceptable justification of the formula. After all, we are basing that guess on just eight or ten values of the infinite sequence a_n, and it is certainly possible that those values happen to follow the pattern we've guessed simply by accident. Maybe the true pattern is much more complicated, and we have been tricked by the small number of cases we have considered. It is necessary to show that the guessed pattern is correct by supplying a logical argument.

Our argument would begin by checking the initial conditions we offered. In other words, we would verify by hand that $a_0 = 1$ and $a_1 = 2$. This serves as a basis for the verification of the recursive formula. Now what we want to do is assume that we have already calculated all the values $a_0, a_1, \cdots, a_k$ for some $k \geq 1$, and **show** that a_{k+1} must equal $a_k + a_{k-1}$. It is very important to understand that *we do not want to compute the value of* a_{k+1}*. We only want to prove* *that* $a_{k+1} = a_k + a_{k-1}$. The major error made doing these types of problems is attempting to compute the specific value of a_{k+1}. **Don't fall for that trap!** After all, if it were possible to actually compute the specific value of a_{k+1}, then we could find a formula for a_n in general, and we wouldn't have to be seeking a recursive relation at all.

Here is how the argument would go in the bit string example. Suppose we have lists of the good bit strings of lengths $0, 1, \cdots, k$. Here is how to make a list of all the good bit strings of length $k+1$. First, take any good bit string of length k and add a 1 on the right hand end. The result must be a good bit string of length $k+1$ (since we added a 1 to the end, and the original bit string didn't have two consecutive 0's, the new bit string cannot have two consecutive 0's either). In that way we form some good bit strings of length $k+1$. In fact, we have built exactly a_k good bit strings of length $k+1$. But wait, there's more! (as they say in those simple-minded TV ads). Another way to build a good bit string of length $k+1$ is to take a good bit string of length $k-1$ and add 10 to the right end. Clearly these will also be good bit strings of length $k+1$. And these all end with a 0, so they are all new ones, and not ones we built in the previous step. How many are there of this type? One for each of the good bit strings of length $k-1$, or a total of a_{k-1}. Thus, so far we have built $a_k + a_{k-1}$ good bit strings of length $k+1$. Now we will show that in fact we have a complete list of all good bit string of length $k+1$, and that will complete the proof that $a_{k+1} = a_k + a_{k-1}$. But before driving that last nail into the coffin, let's look at the steps outlined above for the case $k+1 = 4$.

The previous paragraph essentially provides an algorithm for building good bit strings of length $k+1$ from good bits strings of lengths k and $k-1$. The algorithm instructs us to add 1 to the right end of all the good bit strings of length k and 10 to the right of all the good bit strings of length $k-1$. Applying the algorithm for the case $k+1 = 4$, gives the following list, where the added bits are put in parentheses to make them stand out. 010(1), 011(1), 101(1), 110(1), 111(1), 01(10), 10(10), and 11(10).

There remains one detail to iron out. It is clear that the algorithm will produce good bit strings of length $k+1$. But, does it produce *every* good bit string of length $k+1$? If it does not, then the recursive relation we are offering for the solution to the counting problem will eventually begin to produce answers that are too small, and we will undercount the number of good bit strings. To see that we do count all good bit strings of length $k+1$, consider any particular good bit string of length $k+1$, call it s for short, and look at the right most bit of s. There are two possibilities for that bit. It could be a 1. If that is so, then when the 1 is removed the remaining bit string is a good of length k (it can't have two adjacent 0's since s doesn't have two adjacent 0's). That means the bit string s is produced by adding a 1 to the right end of a good bit string of length k, and so s is produced by the first step in the algorithm. The other option for s is that the right most bit is a 0. But then the second bit in from the right must be a 1, since s is a good bit string, so it doesn't have adjacent 0's. So the last two bits on the right of s are 10. If those two bits are removed, there remains a good bit string of length $k-1$. Thus s is produced by adding 10 to the right end of a good bit string of length $k-1$, and so s is produced by the second case in the algorithm.

In a nutshell, we have shown our algorithm produces $a_k + a_{k-1}$ good bit strings of length $k+1$, and that the algorithm does not miss any good bit strings of length $k+1$. Thus we have proved that $a_{k+1} = a_k + a_{k-1}$ for all $k \geq 2$.

Example 34.1 was explained in excruciating detail. Normally, the verifications will be much more briefly presented. It takes a while to get used to recursive counting, but once the light goes on, the beauty and simplicity of the method will become apparent.

34.2 *Examples*

Example 34.2. *This example is a little silly since it is very easy to write down a formula to solve the counting problem. But the point of the example is not find the solution to the problem but rather to exhibit recursive counting in action. The problem is to compute the total number of individual squares on an $n \times n$ checkerboard. If we let the total number of squares be* *denoted by s_n, then obviously $s_n = n^2$. For example, an ordinary checkerboard is an 8×8 board, and it has a total of $s_8 = 8^2 = 64$ individual squares. But let's count the number of squares recursively. Clearly $s_0 = 0$. Now suppose we have computed the values of $s_0, s_1, \cdots, s_k$, for some $k \geq 0$. We will show how to compute s_{k+1} from those known values. To determine s_{k+1}, draw a $(k+1) \times (k+1)$ checkerboard. (You should make a little sketch of such a board for say $k+1 = 5$ so you can follow the process described next.) From that $(k+1) \times (k+1)$ board, slice off the right hand column of squares, and the bottom row of squares. What is left over will be a $k \times k$ checkerboard, so it will have s_k individual squares. That means that*

$$s_{k+1} = s_k + \text{ the number of squares sliced off}$$

Now ignore the lower right hand corner square for a moment. There are k other squares in the right hand column that was sliced off. Likewise, ignoring the corner square, there are k other squares in the bottom row that was sliced off. Hence the total number of squares sliced off was $k + k + 1$, the 1 accounting for the corner square. Thus

$$s_{k+1} = s_k + k + k + 1 = s_k + 2k + 1$$

So a recursive solution to the problem of counting s_n = number of individual squares on an $n \times n$ checkerboard is

$$s_0 = 0, \text{ and}$$
$$s_{k+1} = s_k + 2k + 1, \text{ for } k \geq 0.$$

Using the recursive relation, we get $s_0 = 0$, $s_1 = s_0 + 2(0) + 1 = 0 + 0 + 1 = 1$, $s_2 = s_1 + 2(1) + 1 = 1 + 2 + 1 = 4$, $s_3 = s_2 + 2(2) + 1 = 4 + 4 + 1 = 9$, and so on, giving what we recognize as the correct answers.

Example 34.3. *Suppose we have available an unlimited number of pennies and nickels to deposit in a vending machine (a really old vending machine it seems, since it even accepts pennies). Let d_n be the number of different ways of depositing a total of n cents in the machine. Just to make sure we understand the problem, let's compute d_n for a few small values of n. Clearly $d_0 = 1$ since there is only one way to deposit no money in the machine* *(namely don't put any money in the machine!).* $d_1 = 1$ *(put in one penny),* $d_2 = 1$ *(put in two pennies),* $d_3 = 1$ *(put in three pennies),* $d_4 = 1$ *(put in four pennies). Now things start to get exciting!* $d_5 = 2$ *(put in five pennies or put in one nickel). And even more thrilling is* $d_6 = 3$ *(the three options are (1) six pennies, (2) one penny followed by a nickel, and (3) one nickel followed by a penny). That last count indicates a fact that may not have been clear: the order on which pennies and nickels are deposited is considered important. With a little more trial and error with pencil and paper, further values are found to be* $d_7 = 4$, $d_8 = 5$, $d_9 = 6$, $d_{10} = 8$, $d_{11} = 11$, *and* $d_{12} = 15$. *It is hard to see a formula for these values. But it is duck soup to write down a recursive relation that produces this sequence of values. Think of it this way, suppose we wanted to put* n *cents in the machine, where* $n \geq 5$. *We can make the first coin either a penny or a nickel. If we make the first coin a penny, then we will need to add* $n-1$ *more cents, which can be done in* d_{n-1} *ways. On the other hand, if we make the first coin a nickel, we will need to deposit* $n-5$ *more cents, and that can be done in* d_{n-5} *ways. By the sum rule of counting, we conclude that the number of ways of depositing* n *cents is* $d_{n-1} + d_{n-5}$. *In other words,* $d_n = d_{n-1} + d_{n-5}$ *for* $n \geq 5$.

Since our recursive relation for d_n *does not kick in until* n *reaches* 5, *we will need to include* d_0, d_1, d_2, d_3, *and* d_4 *as initial terms. So the recursive solution to this counting problem is*

$$d_0 = 1 \qquad d_1 = 1 \qquad d_2 = 1 \qquad d_3 = 1 \qquad d_4 = 1$$
$$\text{for} \quad n \geq 5, \qquad d_n = d_{n-1} + d_{n-5}$$

Example 34.4 (The Tower of Hanoi). *The classic example of recursive counting concerns the story of the Tower of Hanoi. A group of monks wished a magical tower to be constructed from 1000 stone rings. The rings were to be of 1000 different sizes. The size and composition of the rings was to be designed so that any ring could support the entire weight of all of the rings smaller than itself, but each ring would be crushed beneath the weight of any larger ring.*

The monks hired the lowest bidder to construct the tower in a clearing in the dense jungle nearby. Upon completion of construction the engineers brought the monks to see their work. The monks admired the exquisite work-

manship, but informed the engineers that the tower was not in the proper clearing.

In the jungle there were only three permanent clearings. The monks had labelled them A, B and C. The engineers had labelled them in reverse order. The monks instructed the engineers to move the tower from clearing A to clearing C!

Because of the massive size of the rings, the engineers could only move one per day. No ring could be left anywhere in the jungle except one of A, B, or C. Finally each clearing was only large enough so that rings could be stored there by stacking them one on top of another.

The monks then asked the engineers how long it would take for them to fix the problem.

Before they all flipped a gasket, the most mathematically talented engineer came upon the following solution.

Let H_n *denote the minimum number of days required to move an* n *ring tower from* A *to* C *under the constraints given. Then* $H_1 = 1$*, and in general an* n *ring tower can be moved from* A *to* C *by first moving the top* $(n-1)$ *rings from* A *to* B *leaving the bottom ring at* A*, then moving the bottom ring from* A *to* C*, and then moving the top* $(n-1)$ *rings from clearing* B *to clearing* C*. That shows* $H_n \leq 2 \cdot H_{n-1} + 1$*, for* $n \geq 2$*, and a little more thought shows the algorithm just described cannot be improved upon. Thus* $H_n = 2 \cdot H_{n-1} + 1$*.*

Using the initial condition $H_1 = 1$ *together with the recursive relation* $H_n = 2 \cdot H_{n-1} + 1$*, we can generate terms of the sequence:*

$$1, 3, 7, 15, 31, 63, 127, 255, 511, \cdots,$$

and it looks like $H_n = 2^n - 1$ *for* $n \geq 1$*, which can be verified by an easy induction.*

So, the problem would be fixed in $2^{1000} - 1$ *days, or approximately* 2.93564×10^{296} *centuries. Now, that is* job security!

34.3 General rules for finding recursive solutions

Here are a few general rules for solving counting problems recursively:

(1) do a few small cases by brute force,

(2) think recursively: how can a larger case be solved if the solutions to smaller cases are known, and

(3) check the numbers produced by the recursive solution to make sure they agree with the values obtained by brute force.

34.4 Exercises

Exercise 34.1. *On day zero, a piggy bank contains* \$0. *Each day, one more penny is added to the bank than the day before. So, on day* 1, *one penny is added, on day* 2, *two pennies are added. Write a recursive formula for the total number of pennies in the bank each day,* $n = 0, 1, 2, \ldots$.

Exercise 34.2. *Al climbs stairs by taking either one or two steps at a time. For example, he can climb a flight of three steps in three different ways: (1) one step, one step, one step or (2) two step, one step, or (3) one step, two step. Determine a recursive formula for the number of different ways Al can climb a flight of n steps.*

Exercise 34.3. *Find a recurrence relation for the number of bit strings of length n that contain an even number of* 0's.

Exercise 34.4. *Find a recurrence relation for the number of bit strings of length n that contain two consecutive* 0's.

Exercise 34.5. *Find a recurrence relation for the number of bit strings of length n that contain the string* 01.

Exercise 34.6. *Find a recurrence relation for the number of ternary strings of length n that contain two consecutive* 0's.

Exercise 34.7. *Find a recurrence relation for the number of subsets of* $\{1, 2, 3, \ldots, n\}$ *that do not contain any consecutive integers. Examples for* $n = 9$: *the subset* $\{1, 3, 8\}$ *is good, but the subset* $\{2, 5, 6, 9\}$ *is bad since it contains the consecutive integers* $5, 6$.

Exercise 34.8. *Suppose in the original Tower of Hanoi problem there are four clearings A, B, C, D. Find a recursive relation for J_n, the minimum number of moves needed to transfer the tower from clearing A to clearing D.*

34.5 Problems

Problem 34.1. *Suppose on December* 31, 2000, *a deposit of* \$100 *is made in a savings account that pays* 10% *annual interest (Ah, those were the days!). So one year after the initial deposit, on December* 31, 2001, *the account will be credited with* \$10, *and have a value of* \$110. *On December* 31, 2002 *that* *account will be credited with an additional* \$11*, and have value* \$121*. Find a recursive relation that gives the value of the account n years after the initial deposit.*

Problem 34.2. *Sal climbs stairs by taking either one, two, or three steps at a time. Determine a recursive formula for the number of different ways Sal can climb a flight of n steps. In how many ways can Sal climb a flight of* 10 *steps?*

Problem 34.3. *Passwords for a certain computer system are strings of uppercase letters. A valid password must contain an even number of X's. Determine a recurrence relation for the number of valid passwords of length n.*

Problem 34.4. *A (cheap) vending machine accepts pennies, nickels, and dimes. Let d_n be the number of ways of depositing n cents in the machine, where the order in which the coins are deposited matters. Determine a recurrence relation for d_n. Give the initial conditions.*

Problem 34.5. *Suppose the Tower of Hanoi rules are changed so that stones may only be transferred to an adjacent clearing in one move. Let I_n be the minimum number of moves required to transfer tower from clearing A to clearing C?*

(a) By brute force, determine I_1, I_2, and I_3.

(b) Find a recursive relation for I_n.

(c) Guess a formula for I_n.

Problem 34.6. *Find a recurrence relation for the number of binary strings of length n which do not contain the substring* 010.

Problem 34.7. *Find a recurrence relation for the number of ternary strings of length n that contain three consecutive zeroes.*

Problem 34.8. *Find a recurrence relation for the number of quaternary strings which contain two consecutive* 1*'s.*

Problem 34.9. *Let n be a positive integer. Find a recurrence relation that counts the number of increasing sequences of distinct integers that start with* 1 *and end with n. Example: For $n = 4$, there are* 4 *such sequences. They are* $1,4$, $1,2,4$, $1,3,4$, *and* $1,2,3,4$.

Problem 34.10. *Find a recurrence relation that counts the number of ways of painting the squares of an $n \times n$ chess board with blue, green, and yellow paint so that no two squares that share an edge have the same color.*

35
Solutions to Recurrence Relations

In chapter 34, it was pointed out that recursively defined sequences suffer from one major drawback: In order to compute a particular term in the sequence, it is necessary to first compute all the terms of the sequence leading up to the one that is wanted. Imagine the chore to calculate the 250^{th} Fibonacci number, f_{250}! For problems of computation, there is nothing like having a formula like $a_n = n^2$, into which it is merely necessary to plug the number of interest.

35.1 Solving a recursion by conjecture

It may be possible to find a formula for a sequence that is defined recursively. When that can be done, you have the best of both the formula and recursive worlds. If we find a formula for the terms of a recursively defined sequence, we say we have **solved** the recursion.

Example 35.1. *Here is an example: The sequence $\{a_n\}$ is defined recursively by the initial condition $a_0 = 2$, and the recursive formula $a_n = 2a_{n-1} - 1$ for $n \geq 1$. If the first few terms of this sequence are written out, the results are*

$$2, 3, 5, 9, 17, 33, 65, 129, \cdots,$$

and it shouldn't be too long before the pattern becomes clear. In fact, it looks like $a_n = 2^n + 1$ is the formula for a_n.

You have to recognize the slightly hidden powers of 2: $1, 2, 4, 8, 16, 32, 64, \ldots$.

To prove that guess is correct, induction would be the best way to go.

Here are the details. Just to make everything clear, here is what we are going to show: If $a_0 = 2$, and $a_n = 2a_{n-1} - 1$ for $n \geq 1$, then $a_n = 2^n + 1$ for all $n \geq 0$. The basis for the inductive proof is the case $n = 0$. The correct value for a_0 is 2, and the guessed formula has value 2 when $n = 0$, so that checks out. Now for the inductive step: suppose that the formula for a_k is correct for a particular $k \geq 0$. That is, assume $a_k = 2^k + 1$ for some $k \geq 0$. Let's show that the formula must also be correct for a_{k+1}. That is, we want to show $a_{k+1} = 2^{k+1} + 1$. Well, we know that $a_{k+1} = 2a_k - 1$, and hence $a_{k+1} = 2(2^k + 1) - 1 = 2^{k+1} + 2 - 1 = 2^{k+1} + 1$, just as was to be proved. It can now be concluded that the formula we guessed is correct for all $n \geq 0$.

In example 35.1, it was possible to guess the correct formula for a_n after looking at a few terms. In most cases the formula will be so complicated that that sort of guessing will be out of the question.

35.2 *Solving a recursion by unfolding*

There is a method that will nearly automatically solve any recurrence of the form $a_0 = a$ and for, $n \geq 1$, $a_n = ba_{n-1} + c$ (where a, b, c are constants). The method is called **unfolding**.

Example 35.2. *As an example, let's solve $a_0 = 2$ and, for $n \geq 1$, $a_n = 5 + 2a_{n-1}$. The plan is to write down the recurrence relation, and then substitute for a_{n-1}, then for a_{n-2}, and so on, until we reach a_0. It looks like this*

$$\begin{aligned} a_n &= 5 + 2a_{n-1} \\ &= 5 + 2(5 + 2a_{n-2}) = 5 + 5(2) + 2^2 a_{n-2} \\ &= 5 + 5(2) + 2^2(5 + 2a_{n-3}) = 5 + 5(2) + 5(2^2) + 2^3 a_{n-3}. \end{aligned}$$

If this substitution is continued, eventually we reach an expression we can compute in closed form:

In the next to last step we use the formula for adding the terms of a geometric sequence.

$$\begin{aligned}
a_n &= 5 + 5(2) + 5(2^2) + 5(2^3) + \cdots + 5(2^{n-1}) + 2^n a_0 \\
&= 5(1 + 2 + 2^2 + \cdots + 2^{n-1}) + 2^n(2) \\
&= 5\frac{2^n - 1}{2 - 1} + 2(2^n) \\
&= 5(2^n - 1) + 2(2^n) \\
&= 7(2^n) - 5.
\end{aligned}$$

35.3 *Exercises*

Exercise 35.1. *Guess the solution to* $a_0 = 2$*, and* $a_1 = 4$*, and, for* $n \geq 2$*,* $a_n = 4a_{n-1} - 3a_{n-2}$ *and prove your guess is correct by induction.*

Exercise 35.2. *Solve by unfolding:* $a_0 = 2$*, and, for* $n \geq 1$*,* $a_n = 5a_{n-1}$*.*

Exercise 35.3. *Solve by unfolding:* $a_0 = 2$*, and, for* $n \geq 1$*,* $a_n = 5a_{n-1} + 3$*.*

Hint: This one will involve applying the geometric sum formula.

35.4 *Problems*

Problem 35.1. *Guess the solution to* $a_0 = 1$*, and* $a_1 = 5$*, and, for* $n \geq 2$*,* $a_n = a_{n-1} + 2a_{n-2}$ *and prove your guess is correct by induction.*

Problem 35.2. *Solve by unfolding:* $a_0 = 2$*, and, for* $n \geq 1$*,* $a_n = 7a_{n-1}$*.*

Problem 35.3. *Solve by unfolding:* $a_0 = 2$*, and, for* $n \geq 1$*,* $a_n = 7a_{n-1} + 3$*.*