Notes 16

Today's topic is *recurrences.* We've seen recurrences before (ie, Fibonacci numbers, recursive definition of factorial) but now we will study them more methodically. The general objective is to find a solution in *closed form*—that is, one that can be computed directly, without invoking the recursive definition of the recurrence. This is desirable for the same reason as for sums: closed forms make it easier to understand the "shape" of the sum/recurrence, and also do to some things with it (such as inverting the function).

When faced with a new recurrence, there are typically two steps you need to take:

1. First you **solve** the recurrence by coming up with a closed form. This is done by hook or by crook—you can guess, lie, cheat, and steal as long as you get the answer.
2. Then you *prove* that your solution is correct. Because of the recursive nature of recurrences, this is almost always done by induction.

1 Hanoi Towers

We'll start with a simple and famous example of a recurrent problem known as the towers of Hanoi, invented by French mathematician Edouard Lucas in 1883.

In this problem, we are given a collection of disks of different sizes and three pegs. At the beginning, all disks are stocked in decreasing order of size on one peg. Our objective is to move all disks from this peg to another one moving just one disk at a time and never moving a larger disk onto a shorter one.

At first, it is not even clear that this is possible. We are going to give a general method that works for n disks whatever n is (the original problem was about 64 disks) and we will estimate the number of moves needed.

Definition 1.1 Let $T(n)$ be the number of steps needed to move the n-disk tower of the first peg to another peg.

It is easy to check that $T(1) = 1$, $T(2) = 3$ and $T(3) \leq 7$. Now let us consider the following recursive solution to our problem:

- first, move the upper $n-1$ disks to the third peg,

- then move the largest disk to the second peg,
- finally move the $(n-1)$-disk-tower from the third peg to the second peg.

As the first and the third step take $T(n-1)$ moves each and the second step takes one move, this shows that $T(n) \leq 2T(n-1)+1$. Therefore, there exist a solution for all n. However, it could possibly be the case that our recursive algorithm is not optimal. We are going to show that in fact, it is optimal, there is no better way to move the disks.

To see why, notice that at some point we have to move the largest disk, say, from peg 1 to peg 2. The only time when this is possible is when all the other disks are on peg 3 and then they necessarily are in decreasing order of size. But we need at least $T(n-1)$ moves to reach that position and then we need one move to move the largest disk and then $T(n-1)$ more moves to put the (n-1)-disk-tower onto it. Therefore, $T(n) \geq 2T(n-1)+1$, thus $T(n) = 2T(n-1)+1$.

The first step is to solve the recurrence. There are two typical ways to go about this:

1.1 Guess

It is not hard to guess a closed formula for $T(n)$. If we compute some values using the just obtained recursion, we will conjecture that $T(n) = 2^n - 1$. A very easy inductive proof shows that this is indeed correct.

In general this "guess and verify" method is best to solve recurrences. The more practice you have with recurrences, the more often you can come up with a good guess.

1.2 Plug and Chug

If it is not so easy to guess what the correct formula is, one can try the so-called *Plug and Chug* or *Brute Force* method we illustrate here:

$$T(n) = 1+2T(n-1) = 1+2+4T(n-2) = 1+2+4+T(n-3) = \cdots = 1+2+4+\cdots+2^{n-1}$$

The idea here is to keep expanding the recurrence until you see some sort of pattern; this pattern then tells you what sort of guess you should use. Of course, this is only a guess—you still need to prove that your answer is correct.

1.3 Verification

verifying the correctness of our guess can be done by simple induction: the inductive step goes

$$\begin{aligned} T(n) &= 2T(n-1)+1 \\ &= 2(2^{n-1}-1)+1 \\ &= 2^n-2+1 \\ &= 2^n-1 \end{aligned}$$

1.4 Change of Variables

One other useful trick we can apply here is change of variables. If we define $S(n) = T(n)+1$, then

$$\begin{aligned} S(n) &= T(n)+1 \\ &= (2T(n/2)+1)+1 \\ &= 2(T(n/2)+1) \\ &= 2S(n/2) \end{aligned}$$

So it is obvious that $S(n) = 2^n$, which means $T(n) = S(n) - 1 = 2^n - 1$.

Change of variables is often a good way to get read of extra terms (the +1 above) that are making the equation complicated.

2 Linear Recurrences

Hanoi is a particularly simple example of a *linear recurrence*. Here's a more complicated one:

Graduate Student Job Prospects Problem:

Let's assume that due to budgetary constraints, there are at most N faculty positions available. Every year, each professor graduates one student who become a professor upon graduation. Except, first year professor are too busy and don't graduate any students. To make matters worse, everyone lives forever and no one retires.

The question that graduate students ask is: will there be any positions when I graduate? That is, when will all the jobs be filled.

We also impose the boundary condition that in year 1, there is one professor. To solve this problem, we will set up a recurrence.

Let $f(n)$ be the number of professors in year n. The first few values are,

$f(0) = 0$

$f(1) = 1$ (new hire)

$f(2) = 1$ (no students yet)

$f(3) = 2$ (1 old prof, 1 new)

$f(4) = 3$ (2 old, 1 new)

$f(5) = 5$ (3 old, 2 new)

For $n \geq 2$, $f(n) = f(n-1) + f(n-2)$ (first term is number of previous old fculty, second is number of new faculty). The Fibonacci recurrence.

We can now define linear recurrence.

Definition 2.1 A recurrence is *linear* and *homogeneous* if it is of the form

$$f(n) = a_1 f(n-1) + a_2 f(n-2) + \ldots a_d f(n-d) = \sum_{i=1}^{d} a_i f(n-i)$$

for fixed constants a_1, a_2, a_3, $\ldots$, a_d, and d.

Definition 2.2 The *order* of the recurrence is d.

Hanoi is order 1 but *inhomogeneous*: we'll return to this later.

(e.g., the Fibanocci recurrence has order 2)

3 Solving the Recurrence

Let's try the guess and verify method for the Fibonacci recurrence.

As with differential equations, we often start by ignoring the boundary conditions.

Guess: $f(n) = \alpha^n$ for constant α.

Verify:

$$\begin{aligned} & f(n) = f(n-1) + f(n-2) \\ \Longrightarrow \quad & \alpha^n = \alpha^{n-1} + \alpha^{n-2} \\ \Longrightarrow \quad & \alpha^2 = \alpha + 1 \\ \Longrightarrow \quad & \alpha^2 - \alpha - 1 = 0 \\ \Longrightarrow \quad & \alpha = \tfrac{1 \pm \sqrt{5}}{2} \end{aligned}$$

So, we have two possible solutions: $f(n) = \alpha_1^n$ and $f(n) = \alpha_2^n$ where $\alpha_1, \alpha_2 = \frac{1\pm\sqrt{5}}{2}$.

(α_1 is called the *golden ratio*).

In fact, any linear combination is a solution.

Theorem 3.1 If $f_1(n)$ and $f_2(n)$ satisfy a linear homogeneous recurrence (ignoring boundary conditions) then so does $f(n) = c_1 f_1(n) + c_2 f_2(n)$ for arbitrary c_1, c_2.

Proof.

$$f_1(n) \text{ is a solution } \Longrightarrow f_1(n) = \sum_{i=1}^{d} a_i f_1(n-i)$$

$$f_2(n) \text{ is a solution } \Longrightarrow f_2(n) = \sum_{i=1}^{d} a_i f_2(n-i)$$

Thus,

$$c_1 f_1(n) + c_2 f_2(n) = \sum_{i=1}^{d} a_i (c_1 f_1(n-i) + c_2 f_2(n-i))$$

(just multiplying above equations be c_1 and c_2 and adding).

So,

$$f(n) = \sum_{i=1}^{d} a_i f(n-i)$$

■

Corollary 3.2 If $f(n) = \alpha_1^n$ and $f(n) = \alpha_2^n$ are two solutions to a linear recurrence (without boundary conditions), then $f(n) = c_1\alpha_1^n + c_2\alpha_2^n$ is also a solution for any constants c_1 and c_2.

Thus, we know the solution to the Fibonacci recurrence is

$$f(n) = c_1 \left(\frac{1+\sqrt{5}}{2}\right)^n + c_2 \left(\frac{1-\sqrt{5}}{2}\right)^n$$

To determine the constants, we use the boundary conditions.

$$f(0) = 0 = c_1 + c_2$$
$$f(1) = 1 = c_1 \left(\frac{1+\sqrt{5}}{2}\right) + c_2 \left(\frac{1-\sqrt{5}}{2}\right)$$

Thus, $c_1 = 1/\sqrt{5}$ and $c_2 = -1/\sqrt{5}$. Surprisingly, the above formula gives the Fibanocci numbers, which is a non-obvious result. In fact, it isn't even clear the above number is always an integer.

To answer our original question, we want the smallest n such that $f(n) \geq N$. Observe that $\left|\frac{1-\sqrt{5}}{2}\right| < 1$, so any powers of it are also less than 1. For large values of n, this term will becomes very close to zero, and thus the first term converges to the nth Fibanocci number. Even for small values of n, the first term will give us the answer to within 1.

$$\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n \geq N \Longrightarrow n \geq \frac{\log(\sqrt{5}N)}{\log\left(\frac{1+\sqrt{5}}{2}\right)} = \Theta(\log N)$$

For numerical values of N, we can give accurate answers. For $N = 20,000$, we have $n = 21$. In general, we are most interested in asymptotics. In that case, the jobs will be filled in $\Theta(\log N)$ years.

4 Solving General Linear Recurrences

For an arbitrary dth order linear recurrence,

$$f(n) = \sum_{i=1}^{d} a_i f(n-i)$$

Guess: $f(n) = \alpha^n$ for some α

Verify:

$$\begin{aligned}
& f(n) = \textstyle\sum_{i=1}^{d} a_i f(n-i) \\
\Longleftrightarrow\quad & \alpha^n = \textstyle\sum_{i=1}^{d} a_i \alpha^{n-i} \\
\Longleftrightarrow\quad & \alpha^d = \textstyle\sum_{i=1}^{d} a_i \alpha^{d-i} \\
\Longleftrightarrow\quad & \alpha^d - a_1\alpha^{d-1} - a_2\alpha^{d-2} - \ldots a_d = 0
\end{aligned}$$

This last equation is called *the characteristic equation.* In general, this equation will have d roots. If all the roots are distinct, the solution will be

$$f(n) = c_1\alpha_1^n + c_2\alpha_2^n + \ldots + c_d\alpha_d^n$$

We determine the c_i by using the boundary conditions. We need d equations to find a unique solution.

Note: The above derivation does not constitute a proof that $f(n) = \alpha^n$. Rather, it is a proof that $f(n) = \alpha^n$ implies α is a root of the characteristic equation. We can also observe that the equalities are all "if and only if" so we have also shown that any root of the characteristic equation *is* a solution to the recurrence. However, as we shall now see, there can be other solutions.

4.1 Repeated Roots

If the roots are repeated, we need to modify our solutions. If a root, α, is repeated r times, then the solutions look like

$$\alpha^n, n\alpha^n, n^2\alpha^n, \ldots, n^{r-1}.$$

We will verify this for the case $r = 2$. (i.e. $n\alpha^n$ is a solution). Let α be a double root of the characteristic equation, and let $g(x) = (x^d - a_1x^{d-1} - a_2x^{d-2} - \ldots a_d)x^{n-d}$ (that is, the characteristic equation multiplied by x^{n-d}. Then

$$\begin{aligned} g(x) &= (x-\alpha)^2 h(x) \\ \Longrightarrow g'(x) &= 2(x-\alpha)h(x) + (x-\alpha)^2 h'(x) \end{aligned}$$

Thus, $g'(\alpha) = 0$. Therefore,

$$\begin{aligned} g'(\alpha) &= n\alpha^{n-1} - \sum_{i=1}^{d} a_i(n-i)\alpha^{n-i-1} = 0 \\ \Longrightarrow n\alpha^n &= \sum_{i=1}^{d} a_i(n-i)\alpha^{n-i} \end{aligned}$$

which shows that $n\alpha^n$ is a solution.

Example: We have a plant that lives forever. It produces one offspring the first year of its life and nothing in the future. If we start with one new plant in year 1, how many plants will we have in year n?

Recurrence:

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

At year n, we have the plants from last year plus the one new plant for each first year plant.

Boundary conditions: $f(0) = 0$, $f(1) = 1$.

Characteristic equation: $\alpha^2 - 2\alpha + 1 = 0$.

Double root at 1.

Solution: $c_1 + c_2 n$

Using boundary conditions, we get $f(n) = n$.

Recurrences of the form

$$f(n) - \sum_{i=1}^{d} a_i f(n-i) = 0$$

are called homogeneous recurrences. Next, we study inhomogeneous recurrences.

5 Inhomogeneous Recurrences

Inhomogeneous recurrences look like

$$f(n) - \sum_{i=1}^{d} a_i f(n-i) = g(n)$$

The general method is to do the following:

1. Find the homogeneous solution by setting $g(n) = 0$.
2. Find a particular solution.
3. Add the two together and use the boundary conditions to determine the coefficients of the constant factors.

For those who have taken 18.03, this is similar to solving differential equations.

Example: $f(n) = 4f(n-1) + 3^n$; $f(1) = 1$

Homogeneous solution: $f(n) - 4f(n-1) = 0$. $\alpha = 4$. $f(n) = c4^n$

Particular solution: Guess $f(n) = k3^n$

$$\begin{aligned} k3^n &= 4k3^{n-1} + 3^n \\ 3k &= 4k + 3 \\ k &= -3 \end{aligned}$$

Particular solution: $f(n) = -3^{n+1}$

Combine solutions: $f(n) = c4^n - 3^{n+1}$.

Boundary conditions imply $f(n) = \frac{5}{2}4^n - 3^{n+1}$.

Another example: $f(n) = 4f(n-1) - 4f(n-2) + n$; $f(0) = 4,\ f(1) = 7$

Homogeneous solution: $f(n) - 4f(n-1) + 4f(n-2) = 0$. $\alpha = 2, 2$ (double root). $f(n) = c_1 2^n + c_2 n 2^n$

Particular solution: Guess $f(n) = an + b$

$$\begin{aligned} an + b &= 4(a(n-1) + b) - 4(a(n-2) + b) + n \\ an + b &= n + 4a \end{aligned}$$

Thus, $a = 1,\ b = 4$.

Particular solution: $f(n) = n + 4$

Combine solutions: $f(n) = c_1 2^n + c_2 n 2^n + n + 4$.

Boundary conditions imply $f(n) = n2^n + n + 4$.

When guessing a particular solution, it makes sense to guess a solution that is the same type as $g(n)$. If $g(n)$ is an r-degree polynomial, guess an r-degree polynomial (i.e., $ax^r + bx^{r-1} + \ldots + d$). If $g(n)$ is an exponential, guess an exponential(e.g., if $g(n) = k^n$, guess ck^n).

6 Merge Sort

One of the most popular sorting algorithms is as follows:

To sort n items $x_1, x_2, \cdots, x_n$

- sort $x_1, x_2, \cdots, x_{n/2}$ and $x_{n/2+1}, x_{n/2+2}, \cdots, x_n$
- merge the two lists together.

To merge two lists, simply compare the smallest items of the two lists and put the smaller at the start of the merged list and keep the other one at the top of its short list. Then do it again with the remains of the two lists. Do it as long as there are elements which haven't been put on the new merged list yet.

Definition 6.1 Let $T(n)$ be the number of comparisons needed to sort n numbers.

To estimate $T(n)$, let us note that it takes $2T(n/2)$ steps to sort the two lists of length $n/2$ and $n-1$ steps (at worst) to merge the two lists together. Therefore, $T(n) = 2T(n/2) + n - 1$.

Applying the plug and chug method we get that

$$T(n) = (n-1) + 2T(n/2) = n - 1 + n - 2 + 4T(n/4) + \cdots + n - 2^i + 2^i T(n/2^i)$$

(note that this expansion can be formally verified by induction on i), where $T(n/2^i) = 0$ for $i = log_2(n)$ (all logarithms appearing in subsequent expressions are assumed to be base 2).

Thus

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

The reader should verify by induction that the estimate proven for $T(n)$ is indeed correct.

In this case, it is more convenient to verify in summation form than in closed form. However, can compute the closed form...

6.1 Integer Breakage

You might notice that there's a problem with the above analysis when n is not a power of 2: at some point, the argument to a recursive invocation of the recurrence will become *fractional*, thus outside the domain of our recurrence. So the above analysis only works when n is a power of 2.

It is more correct to write the above recurrence as $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n$. But this makes the analysis messy. So we ignore floors and ceilings and get the previous recurrence.

Why does this not matter? Well, it is easy to prove (by induction) that $T(n)$ is an increasing function of n—that is, $T(n+1) \geq T(n)$. So consider an arbitrary n. Let k be the largest power of 2 less than n, so $k < n \leq 2k$. Then we have just argued that $T(k) \leq T(n) \leq T(2k)$.

Thus $\begin{array}{l} T(k) \leq T(n) \leq T(2k) \\ k\log k - 1 \leq T(n) \leq 2k\log 2k - 1 \end{array}$

It follows that $T(n) = \Theta(k\log k) = \Theta(n\log n)$. So asymptotically, we can ignore the fact that non-powers of two suffer "breakage" into fractions in the recurrence.

This kind of argument is true of almost every recurrence one encounters in computer science, so we usually ignore the integer breakage phenomenon, assume the right "power of" feature to make the analysis easy, and state without proof that our results hold for all n.

7 Divide and Conquer recurrences

Let $T(n) = aT(n/2) + bn$ and $T(1) = 1$, where a and b are constant. We will examine the effect of replacing 2 by the scalar a. When we solve this recurrence by plug and chug expansion, we obtain that

$$T(n) = bn\left(1 + \frac{a}{2} + \left(\frac{a}{2}\right)^2 + \cdots + \left(\frac{a}{2}\right)^{\log n - 1}\right) = bn \sum_{i=0}^{\log n - 1} \left(\frac{a}{2}\right)^i.$$

(recall that logs are base 2). If $a = 2$, the sum is just $(1 + \log n)$. If $a \neq 2$, the sum is a geometric series, with the first or last term dominating depending on whether $a < 2$ or $a > 2$. Plugging in our close for for geometric series, we find that

- for $a < 2$, $T(n) = \frac{2bn}{2-a} + 1 - \frac{2b}{2-a} = \Theta(n)$,
- for $a = 2$, $T(n) = bn \log n = \Theta(n \log n)$, and
- for $a > 2$, $T(n) = \frac{ab}{a-2} n^{\log a} = \Theta(n^{\log a})$.

To verify that the above expressions are correct, we must do induction on n. Here, we prove the result for $a < 2$. The proof for the other cases is similar.

$$T(n) = aT(n/2) + bn = a\frac{2bn}{2(2-a)} + bn = \frac{abn + bn(2-a)}{2(2-a)} = \frac{2bn}{2-a}.$$

Thus we can conclude that the recurrence is very sensitive to any changes in a, but it varies only linearly if we change b. Hence, if we don't care about constant factors, we can write $T(n) = aT(n/2) + \Theta(n)$, with $\Theta(1) = 1$, but we cannot write $T(n) = \Theta(T(n/2))$ without losing a lot of sharpness in our estimate.

As in the previous paragraph, people often use asymptotic notation when they write recurrences. While doing this, it is easy to make false proofs by mixing big O and induction. The key to avoid these mistakes is to make sure that your induction hypothesis makes sense, that is, that it states something meaningful on a number n. For example, it is meaningless to say that "for all n, $T(n) = O(n)$, since $T(n)$ is just a number, not a function, whereas $O(n)$ denotes a class of functions. The correct way to state the above assertion is: "for all n, we have $T(n) \leq cn$. As a general advice, keep big O out of inductive proofs!

Now let us consider the following rather ugly recurrence:

$$S(n) = S(\lfloor n/2 \rfloor) + S(\lceil n/2 \rceil) + 1$$

and let $S(1) = 0$. It would be tedious to expand this recurrence. Therefore, we solve a recurrence which seems to be very similar to this one, and then verify that the result indeed fulfills the above recurrence, too. Namely, first we easily solve $S(n) = 1 + 2S(n/2)$, $S(1) = 0$ and get $S(n) = n - 1$. Then we guess that this solution satisfies the previous recurrence as well, and check that statement by induction, finding that our guess is correct.

Note that this recurrence looked pretty similar to that of the Towers of Hanoi and there was a tremendous difference in the results. This warns us that it is not always easy to get a good guess, especially if we are unexperienced.

So far we have seen a lot of examples on why induction is a great way to prove theorems. Now let's see some of its limitations. Let us consider again the last recurrence, that is, $S(n) = 1 + 2S(n/2)$, and try to prove by induction that $S(n) < n$.

- $S(1) = 0 < 1$
- $S(n) = 1 + 2S(n/2) = 2(n/2) + 1 \leq n + 1$

which is not enough, as we need $S(n) < n$.

This doesn't mean that the result is not true, it is straightforward to prove by induction that $S(n) = n - 1$, it just means that we couldn't establish it this way, as the choice of our claim was wrong, *it was not strong enough.* This shows that sometimes it is easier to prove a stronger statement by induction than a weaker one.

Once again, we point out the importance of checking the base case. For example, here we could "deduce" the induction step of the false claim $S(n) < n - 2$, but it would not be a proof as this claim doesn't hold for $n = 1$.

8 Recurrences in a product form

Consider the recurrence $T(n) \leq nT(\sqrt{n})^2$. As we deal with a product, it is convenient to take logarithms to get

$$\log T(n) \leq 2 \log T(\sqrt{n}) + \log(n).$$

Next, set $S(n) = \log T(n)$ to get

$$S(n) \leq 2S(\sqrt{n}) + \log(n).$$

Expanding this we have $S(n) = O(\log(n) \cdot \log\log(n)$, which yields

$$T(n) = 2^{S(n)} = 2^{O(\log n \cdot \log\log n)} = n^{O(\log\log n)}.$$