Setting up and solving a recurrence relation.

Dan Gusfield

In the towers of Hanoi problem you are given three pegs. On the first peg there are n rings of different diameters stacked according to their diameter, the largest on the bottom. The object is to move the rings to the third peg by moving one ring at a time, with the complication that you can never place a larger ring on top of a smaller ring.

The following is the tower of Hanoi procedure for moving n disks from pole r to pole s.

```
Procedure P(n,r,s)

if n=0 return;

else
begin
P(n-1,r,6-r-s)
move one disk from pole r to pole s.
P(n-1,6-r-s,s)
end

return
```

Actual code (try it out) for this is:

```
#include<stdio.h>

void th(int n, int s, int dest, int * pmoves);

main()
{
int * pmoves;
int n, s, dest;
printf("how many disks to move?\n");
scanf("%d", &n);
printf("enter the source and the final destination poles\n");
scanf("%d%d", &s, &dest);
printf("the source pole is %d, and the destination pole is %d\n", s, dest);
if (s == dest) printf("No go\n");
else {
*pmoves = 0;
th(n, s, dest, pmoves);
printf("The total number of moves is %d\n", *pmoves);
}
}

void th(int n, int s, int dest, int * pmoves)
{
if (n == 0)
  return;

 else {
 th(n-1, s, 6-(s+dest), pmoves);
 printf("move one disk from pole %d to pole %d\n", s, dest);
 *pmoves = 1+*pmoves;
 th(n-1, 6-(s+dest), dest, pmoves);
}
}
```

Now we want to establish the number of moves that this procedure specifies, as a function of n. To do that, we set up a recurrence relation that expresses the number of moves specified by the procedure, as a function of n. We let $T(n)$ denote that number.

Then $T(n) = T(n-1) + 1 + T(n-1) = 2T(n-1) + 1$. and $T(0) = 0$.

That is a recurrence relation for $T(n)$ because T appears on both sides. We would like a closed form solution to $T(n)$, i.e., one that gives a function of n on the right without using T itself on the right.

In class we discussed two methods to find the closed form solution for $T(n)$.

Method 1 You might run the program for a few small values of n, and then look at the output to guess that $T(n) = 2^n - 1$. Having made that guess (claim), you should try to prove that it is right. How? By induction, of course.

basis: n = 1. the program outputs one move, and $2^1 - 1 = 1$, so the case of n = 1 is verified.

I.H. Suppose the claim is true for n up to some value (call it k). I.S. Want to prove that $T(k+1) = 2^(k+1) - 1$. By the R.R. $T(k+1) = 2T(k)+1$. By the I.H., we substitute $2^k - 1$ for $T(k)$, so $T(k+1) = 2 \times (2^k - 1) + 1 = 2^{k+1} - 1$ which completes the proof of the I.S.

Method 2: Unwrapping and drawing a recurrsion tree

In this approach it is good to fix a small value for n (what that should be varies with the particular problem – we will see several examples of this approach where we pick different n). For this problem, say n is 8. We want to know what $T(8)$ is. I don't know, but I do know it is $2 \times T(7) + 1$. I keep a list of the contributions to the total value of $T(8)$. By the recurrence relation, that list starts out with 1.

So we have one node to represent $T(8)$ and by its side, the contribution of 1, and two children nodes, each representing a single value of $T(7)$.

We would like to know what $T(7)$ is, but we don't know. However, we know each $T(7)$ is $1 + 2 \times T(6)$, so those two $T(7)$'s contribute 1 each, for a total of 2, to the total value of $T(8)$.

Continuing in this unwrapping of the recurrence relation, we get binary tree. We label the levels of the tree in order starting from the root, which is labeled level 0. So at level i, there are 2^i nodes, each of which contributes 1 to the total value of what $T(8)$ is. The levels (where something non-zero is contributed to the sum) range from 0 to 7.

Hence, $T(8)$ is the sum of what each level contributes, which is $\sum_{i=0}^{i=7} 2^i$. We know how to solve that sum. It is $2^8 - 1$.

Having greased our intuition by unrwapping for a specific value of n, we go back through the process to see what would change if we use the symbolic n instead of the specific value 8. What any level i contributes to the sum would not change, it is still 2^i no matter what n is, but the number of levels changes to $n - 1$. So in general, this establishes that

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

Summarizing the unwrapping approach, we set up a tree where the root represents $T(n)$. We use the recurrence relation to see what constant amount is contributed at this level to the total. The next level represents the unwrapping of the recurrence one time. The purpose of this unwrapping is to answer two questions: How much does each level i contribute to the sum for $T(n)$; how many levels are there in the full unwrapping that contribute to the sum?. Given the answers to those two questions, we set up a sum, and solve it (with our deep understanding of elementary sums). That solution is then a closed form solution for $T(n)$.

In the next lecture we will use this approach to solve $T(n) = 2T(n/2)+n$, $T(0) = 0$, and also $T(n) = 4T(n/3) + 5n$. You might try the first one before class.

Another example of unwrapping:

$$T(n) = 4T(n/3) + n/2,$$
$$T(1) = 450.$$

In general it helps to recall the geometric sum, that

$$\sum_{i=0}^{k} r^i = (r^{k+l} - 1)/(r - 1).$$

One way to solve the recurrence is to unroll the recurrence like this:
$T(n) = 4T(n/3) + n/2 = 16T(n/9) + 4(n/6) + n/2...$

In general, the additive part of level i (we start counting levels from zero) is: $4^i \times \frac{n}{3^i \times 2}$. $T(n)$ is gotten by adding in those terms - there are $\log_3 n$ of them (why?) and then adding in the $450n$ that comes from the base case.

So unwrapping in this way gives that

$$T(n) = 450n + n/2 \times \sum_{i=0}^{\log_3 n} (4/3)^i.$$

Now apply the geometric sum to get:

$$T(n) = 450n + n/2 \times 3((4/3)^{(\log_3 n)+1} - 1).$$

So

$$T(n) = 450n + n/2 \times (4(4/3)^{(\log_3 n)} - 3).$$

Now the question is what to do with the $(4/3)^{(\log_3 n)}$. But that is the same as $n^{\log_3 (4/3)} = n^{(\log_3 4)-1}$. So $T(n) = 450n + n/2 \times (4n^{(\log_3 4)-1}) - 3)$. So $T(n) = 450n + (2n^{\log_3 4}) - 3n/2$. By the way, that result is $\theta(n^{\log_3 4})$.

More on unwraping to solve recurrence relations.

Consider the recurrence $T(n) = 2T(n/2) + n$, with the base condition $T(1) = 0$. We unwrap the recurrence, and display the unrwrapping in the form of a tree. The root of the tree represents $T(n)$. We don't know what $T(n)$ is but we do know that for any n (a power of two), $T(n)$ is given by the sum of certain numbers. To start, it is the sum of n and twice $T(n/2)$. So we put n off to the side in a column that will be added up, and draw two children of the root, each labeled with $T(n/2)$. So if we only knew what $T(n/2)$ is, we would be done. But we don't. All we know is that each $T(n/2)$ contributes $n/2$ to the sum, and has two children, each labeled with $T(n/4)$. In general, each level i in the tree (counting from the root, which is labeled 0) has exactly 2^i nodes, each of which contributes $n/(2^i)$, and so each level contributes n to the sum. Since n falls by a half at each level, there are $log_2 n$ levels that contribute to the sum, hence the total sum is $nlog_2 n$. That is $T(n) = nlog_2 n$.

Note that if the base case was $T(1) = 5$, then the close form solution would be $T(n) = nlog_2 n + 5n = \theta(nlogn)$. (Why didn't I write the base of the logarithm inside the parentheses? Answer: because of the θ notation and the fact that $log_a n = c(a,b) log_b n$ where $c(a,b)$ is a constant that depends only on a and b.

Now we consider the more complex recurrence: $T(n) = 5T(n/4) + 3n^2$, $T(1) = 0$.

If you unwrap with a specific number for n, you should use one that is a power of 4. But here, we will unwrap for a general n. The tree branches five ways at each node, and it has $log_4 n$ levels that contribute to the sum. Hence each level i contributes $5^i \times 3(n/4^i)^2 = 3n^2(5/16)^i$. Hence

$$T(n) = 3n^2 \sum_{i=0}^{i=(\log_4 n)-1} (5/16)^i.$$

Let us ignore the $3n^2$ part for now and concentrate on the sum. Applying the geometric sum, the sum is

$$[(5/16)^{\log_4 n} - 1]/(5/16 - 1).$$

Here we can use the fact that $x^{log_y n} = n^{log_y x}$. We use that fact twice, once for $x = 5$ and once for $x = 16$, so that the sum is

$$[(n^{\log_4 5}/n^2) - 1]/(-11/16).$$

Now we can multiply by the $3n^2$ which is on the outside of the sum. The result is

$$(3n^{\log_4 5} - 3n^2)/(-11/16) = (3n^2 - \log_4 5)/(11/16) = \theta(n^2).$$

The Master Method

In order to read this, review the discussion of the notation O, Ω, and θ in the book.

For recurrences of the form $T(n) = aT(n/b) + f(n)$ the following three rules encapsulate the unwrapping ideas that we have illustrated above, and apply to most (but not all) of the recurrence relations one will encounter in computer science:

1. If $f(n) = O(n^{(\log_b a)-\epsilon})$, for some $\epsilon > 0$, then $T(n) = \theta(n^{(\log_b a)})$. Notice that we don't even mention the base case of the recurrence.

2. If $f(n) = \theta(n^{(\log_b a)})$ then $T(n) = \theta(n^{(\log_b a)} \times logn)$.

3. If $f(n) = \Omega(n^{(\log_b a)+\epsilon})$, for some $\epsilon > 0$, and $af(n/b) \leq cf(n)$ for some $c < 1$ and every n greater than some n_0 (which you get to pick), then $T(n) = \theta(f(n))$.

By symbolically unwrapping the recurrence, and applying the geometric sum etc. you should be able to prove rules 1 and 2 for n a power of b. That will be part of the next homework assignment.