MATH 22

Lecture D: 9/11/2003

CATALAN NUMBERS

for life's not a paragraph
And death i think is no parenthesis.
—e. e. cummings, *Since feeling is first*

All animals are equal, but some
animals are more equal than others.
—George Orwell, *Animal Farm*

Administrivia

- No office hours today (plane at 7:30!)
- Left over: Why is the Twin Prime conjecture so difficult?
- Academic Resource Center for extra help: http://ase.tufts.edu/arc
- http://denenberg.com/LectureD.pdf
- Project 2 handed out, due Tuesday

  A correction: In #4 just do the case $n=10$, not $n = 1, 2, \ldots, 10$

Today: Lots o' stuff left over from last lecture; Catalan numbers and their applications; maybe finish early for review

Parentheses

How many ways can we write n pairs of parentheses?

This is easy: We need to write down $2n$ symbols, of which n are left parentheses (which we call "*open*") and n are right parentheses (which we call "*close*"). So, of the $2n$ symbols, we just need to choose which are the opens, and the closes take care of themselves.

The answer is clearly $C(2n, n)$.

E.g. For n=1 we get C(2,1) = 2, namely () and)(.

For n=2 we get C(4,2)=6, namely

(()) ()() ())( )(())()( ))((

(What happens for n=0?)

Now, how many ways can we write n pairs of parentheses such that the result "makes sense"?

For n=1 only () makes sense, for n=2 only the first two of the ways written above make sense.

We prove the following: The number of *bad* arrangements of parentheses is $C(2n, n–1)$.

Therefore the number of *sensible* parenthesizations is

$$C(2n, n) - C(2n, n–1)$$

When Parentheses Go Bad

Here's how we count the bad parenthesizations: We consider *all* strings that contain $n-1$ opens and $n+1$ closes (making a total of $2n$ parentheses). Obviously there are $C(2n, n-1)$ such strings by the usual argument: choose $n-1$ places to put opens out of $2n$ total positions.

We will show that the number of strings with $n-1$ opens and $n+1$ closes is the same as the number of strings with n opens, n closes, and bad parenthesization. Once we do this, the proof is finished.

)Pause to fully comprehend and believe the strategy.(

Our goal is to prove that two sets have the same size. We've done this before, but I'm going to be much more careful (i.e., nitpicking) here, just so everyone appreciates what's involved, and because I'm not happy with Grimaldi's proof! To review, we want to prove that the following two sets have the same size:

X, the set of *badly-parenthesized* strings that contain n opens and n closes (thus with a total length $2n$).

Y, the set of *all* strings that contain $n-1$ opens and $n+1$ closes (again, total length $2n$).

Proving Set Equivalence

How do we show that two sets X and Y have the same size? The typical way is to show how to take any x from X and construct from it some y in Y, such that from different xs we get different ys, and such that every y is constructed from some x or other.

If we do this, then X and Y can be matched up one-to-one with nothing left over. *Each of the requirements must be fulfilled*, otherwise the sets might have different sizes! (We'll study this whole situation later, more abstractly.)

As I said, we've done this informally several times in previous lectures, but without being this careful to touch each base.

Here's another way to do it, just as good:

- Show that from each x we can construct a y, such that different xs lead to different ys, and
- Show that from each y we can construct an x, such that different ys lead to different xs.

In the current case, we must:

- Show how to take a badly-parenthesized string containing n opens and n closes, and construct from it some string with $n-1$ opens and $n+1$ closes.
- Show how to take any string of $n-1$ opens and $n+1$ closes, and construct from it a badly-parenthesized string with n opens and n closes.

In each case we must show that if we start with different strings then the strings we construct are also different.

Construct a *y* from an *x*

We start with a badly-parenthesized string containing exactly *n* opens and *n* closes. (( ) (( ))) ( )) (

What does "badly-parenthesized" really mean? It means that at some point there are more closes than there have been opens!)Pause to appreciate this point.(

Here's the construction: Find the first place where there's one close too many, that is, a close that doesn't have an open. Starting just *after* that close, flip all opens to closes and all closes to opens. (( ) (( ))) (( )

What's in the resulting string? Well, the portion we didn't flip has one extra close. So the portion we did flip has one extra open. That is, one extra open has become a close, so the final resulting string has one extra close; it has $n-1$ opens and $n+1$ closes, as claimed.

Do two different starting strings produce two different ending strings? There are two cases. (1) If two strings are the same up to the first extra close but then differ, they'll differ after the flipping since the same chunk of each will be flipped. (2) Otherwise, the two strings must differ before the first point where *either* has an extra close (why?). In this case they'll still differ after the flipping since we don't touch the part where they differ! (This proof is perfectly valid, if a little *handwaving*.)

Construct an x from a y

This time we start with a string that has $n-1$ opens and $n+1$ closes.

This second construction is actually identical to the first: Starting just *after* the first spot where there's an extra close, flip all the closes to opens and all the opens to closes. The only slight difference is in a justification: How do we know that such a spot exists? In the previous construction, there had to be such a spot because of the bad parenthesization. Here, it's because we know that there are more closes in the starting string.

What's in the resulting string? Say that the part we didn't flip has k opens and $k+1$ closes (we know it has one extra close). So the part we flipped had $n-k-1$ opens and $n-k$ closes *before* the flip, hence $n-k-1$ closes and $n-k$ opens *after* the flip. Adding, we see that the whole string after the flip has n opens and n closes. And it's certainly badly parenthesized since the part we didn't touch was itself badly parenthesized!

Finally, do different starting strings yield different ending strings? Since this construction is identical to the other, the proof is exactly the same.

Catalan Numbers

The numbers

$$C_n = C(2n, n) - C(2n, n-1)$$

are called the *Catalan Numbers* after a Belgian mathematician of the early 19th century.

Fundamental identity:

$$C_n = C(2n, n) - C(2n, n-1) = C(2n,n) / (n+1)$$

Unenlightening proof: Massage one side algebraically into the other. (You will be doing this.)

The identity, rephrased: Of all the possible ways of writing down n pairs of parentheses, exactly one in $n+1$ of these ways makes sense.

Proof that promotes understanding: Arrange the $C(2n,n)$ ways of writing n pairs of parentheses into clumps. Each clump should have exactly one sensible parenthesization and exactly n bad parenthesizations. Thus (a) the number of clumps is the answer we want, and (b) each clump has exactly $n+1$ things in it.

The answer then is clearly $C(2n,n) / (n+1)$.

Regrettably, I don't know a way of making these clumps. So the unenlightening proof is all we've got.

Uses of the Catalan Numbers

The text describes other sets that are counted by the Catalan numbers, e.g.

- The paths from point (0,0) to point (n,n) where each step is either *RIGHT* or *UP*, and where the path never goes above the line from (0,0) to (n,n)
- The ways of arranging a sequence of +1 and –1 such that there are an equal number of each and such that the sum at any point is nonnegative

Here's another excellent one (non-CS students can skip):

How many ways can a string be rearranged using only a stack? That is, suppose input strings are transformed into output strings using *only* these two operations:

- Read the next input character and push it on the stack
- Pop the character on top of the stack and output it

For example, I can take the string ABCD and change it into BACD (push, push, pop, pop, push, pop, push, pop) or CDBA (push, push, push, pop, push, pop, pop, pop) but there's no way I can make it into CABD or ADBC.

It turns out that of the $n!$ possible permutations of an n-character string, exactly C_n of them can be achieved using only a stack. (In the example, I can get to $C_4 = 14$ of the 24 permutations of ABCD using only a stack.) Why?