COMPSCI 575/MATH 513 T

Combinatorics and Graph Theory

Lecture #23: Recurrences
(Tucker Sections 7.1 and 7.2 (with a bit of 6.5))
David Mix Barrington
4 November 2016

Recurrences

- Highlights of Section 6.5
- Some Common Recurrences
- Stairs and Fibonacci Numbers
- Some Examples
- Placing Parentheses
- Systems of Recurrences
- Divide and Conquer

Section 6.5 of Tucker

- I'm mostly skipping this section, but there are two tricks for generating functions we ought to take note of.
- If g(x) is the sum over r of $a_r x^r$, what happens when we take the **derivative** of g(x)? By the rules we learned in a calculus course, $g'(x)$ is the sum over r of $r a_r x^{r-1}$. This makes $xg'(x)$ the sum over r of $r a_r x^r$.

Section 6.5 of Tucker

- We've taken the GF for the sequence $a_0, a_1, a_2, \ldots$ and made a GF for the sequence $0, a_1, 2a_2, 3a_3, \ldots$, multiplying the entry a_r by r.
- Starting from the GF for $1,1,1,\ldots$ which is $1(1\text{-}x)$, we can make GF's for any fixed polynomial in r by using this trick to get a GF for any desired fixed power r^n.

Section 6.5 of Tucker

- The other trick in this section is to take a GF for $a_0, a_1, a_2, \ldots$ and make a GF for $s_0, s_1, s_2, \ldots$, where each s_r is the sum for i from 0 to r of a_i.
- All we need to do is take $g(x) = a_0+a_1x+a_2x^2+\ldots$ and divide it by $1-x$ to get a new series $h(x)$.
- Letting $f(x) = 1/(1-x)$, our new $h(x) = g(x)f(x)$, and each coefficient h_r is just the sum over all i from 0 to r of a_i times 1, since the x^{r-i} coefficient of $f(x)$ is just 1.

What is a Recurrence?

- A **recurrence** is a definition of a sequence a_0, a_1, a_2,… where we define a_0 directly and define all other values a_r in terms of zero or more a_j's, where each j is less than r.
- By induction, each value a_r is properly defined. The base case of r=0 is given since a_0 is defined, and if we know all a_j for $j < r$, the definition gives us a value for a_r.

Some Common Recurrences

- A **linear recurrence** defines a_n as a linear function of the most recent a_j's. The general form is $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_r a_{n-r}$. If $r = 1$ this is just a **geometric series**, with $a_n = a_0(c_1)^n$. For larger r, we need r base cases $a_0, \ldots, a_{r-1}$ rather than just a_0.
- The **Fibonacci recurrence**, defined by $f_0 = 0$, $f_1 = 1$, and $f_n = f_{n-1} + f_{n-2}$ for $n > 1$, is an example of a linear recurrence with $r = 2$.

Some Common Recurrences

- A recurrence might include a fixed function of n as well as the previous values a_j. For example, we could have $a_n = ca_{n-1} + f(n)$, for any function f.
- We could mix up the values, as in the recurrence $a_n = a_0a_{n-1} + a_1a_{n-2} + \ldots + a_{n-1}a_0$.
- Or we could define a two-dimensional recurrence, by a rule like $a_{n,m} = a_{n-1,m} + a_{n-1,m-1}$. (Recognize this one?)

Recurrences in General

- A recurrence fully defines a function, and may or may not give the most efficient way to compute an arbitrary value a_n.
- A **closed form** for a recurrence is a function of n only, not referring to other values of the sequence. We will see a number of general techniques for finding closed forms of recurrences, and for finding recurrences for functions given by closed forms.

Stairs and Fibonacci Numbers

- Suppose an elf is ascending a staircase of r steps, and in one jump it can move one or two steps. Define a_r to be the number of different ways it can reach the top.
- Clearly $a_0 = a_1 = 1$, and $a_2 = 2$ as it could take the steps one by one or both at once.
- In general $a_r = a_{r-1} + a_{r-2}$, since the last jump must come from either step r-1 or step r-2.
- The sequence goes 1, 1, 2, 3, 5, 8, 13, 21, 34, …

Example: Cutting Pizzas

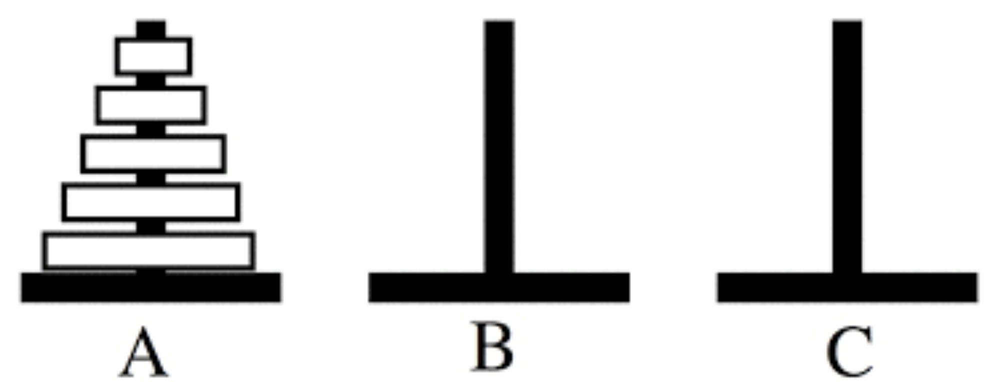
- In COMPSCI 250 we look at the following problem as an example of induction: What is the maximum number of pieces you can get from a convex pizza by n straight-line cuts?
- We can never do better than doubling the number, so $a_0 = 1$, $a_1 = 2$, and $a_2 = 4$ are clearly optimal. But eight pieces with three cuts doesn't appear possible (in the plane), so we need an argument to prove that $a_3 = 7$.

Example: Cutting Pizzas

- The n^{th} straight cut can pass through at most n existing pieces, because it can cross each of the earlier cuts at most once.
- This gives us a rule that $a_n \leq a_{n-1} + n$, and in fact $a_n = a_{n-1} + n$ is achievable if the cuts are in **general position**.
- From the base case of $a_0 = 1$, we can solve this recurrence to get $a_n = (n^2+n+1)/2$.

Example: Tower of Hanoi

- In this puzzle we want to move the n disks from A to C, moving one disk at a time and never putting a larger disk on top of a smaller.
- There is a recursive solution that moves n disks by twice moving n-1 disks, and adding one more move. We get a recurrence $a_0 = 0$, $a_n = 2a_{n-1} + 1$, which solves to $a_n = 2^n - 1$.



Examples From Finance

- If we put \$1000 per year into a savings account that pays 2% annual interest, our balance after n years is given by the recurrence $a_0 = 0$, $a_n = (1.02)a_{n-1} + 1000$.
- If I borrow \$300,000 at 4% annual interest and pay back \$2000 per month, my balance is given by $a_0 = 300000$ and, for all $n > 0$, $a_n = (1+(0.04)/12)a_{n-1} - 2000$. Normally the monthly payment is chosen to make $a_n = 0$ when n is some fixed number of months.

Counting Revisited

- Our familiar counting functions can each be alternatively defined by recurrences:
- Sequences of length r from n choices: $a_0 = 1$, $a_r = na_{r-1}$.
- Permutations: $P(n, n) = 1$, $P(n, r) = nP(n-1, r-1)$.
- Combinations: $C(n, 0) = C(n, n) = 1$, $C(n, r) = C(n-1, r) + C(n-1, r-1)$, where the last equation follows from analyzing the cases of the first item being in or out of the set.

Messier Counting

- Suppose we put n identical balls into k distinct boxes, with between 2 and 4 balls per box. We can model this with a recurrence $a_{n,k} = a_{n-2,k-1} + a_{n-3,k-1} + a_{n-4,k-1}$ with the correct initial conditions. We also know how to solve this problem with generating functions.
- What if the balls come in three colors? Now we have a recurrence $a_{n,k} = 6a_{n-2,k-1} + 10a_{n-3,k-1} + 15a_{n-3,k-1}$ when we take into account the number of ways to fill boxes with 2, 3, or 4 balls.

Placing Parentheses

- If I have a product of n items, how many ways are there to parenthesize them as binary products? Let a_n be this number, so $a_2 = 1$ and $a_3 = 2$ for $(x_1x_2)x_3$ and $x_1(x_2x_3)$.
- The general rule is that a_n is the sum, over all i, of a_ia_{n-i}. This is because the last of our multiplications must be between the product of the first i items and the product of the last n-i. (So we need to define a_1, and the value that makes sense is $a_1 = 1$, along with $a_0 = 0$.)

Systems of Recurrences

- Sometimes we need more than one recurrence to solve a counting problem.
- Consider strings over {a,b,c} with an even number of b's and an odd number of c's.
- If $f(n)$ is the number of such strings of length n, we have that $f(n) = f(n-1) + g(n-1) + h(n-1)$, where $g(n)$ is the number with odd numbers of both b's and c's, and $h(n)$ the number with even numbers of each.

Systems of Recurrences

- Then g(n), for example, is g(n-1) + f(n-1) + i(n-1), where i(n) is the number with an even number of b's and an odd number of c's.
- Each of the four functions is defined by a recurrence using itself and two of the others.
- By induction on n, assuming we define f(0), g(0), h(0), and i(0) to each be 1, we have well-defined and correct values f(n), g(n), h(n), and i(n) for each n.

Divide and Conquer

- Many algorithms take a divide and conquer approach, reducing a problem to similar problems with smaller parameters. Much of COMPSCI 311 is spent analyzing the resources used by such algorithms, and recurrences are a key tool in this analysis.
- If a_n is the number of steps to solve a problem of size n, we often get a recurrence of the form $a_n = ca_{n/2} + f(n)$, where c is a constant and f(n) is the time to split and merge the subproblems.

Simple D&C Examples

- If $c = 1$ and $f(n)$ is constant, we have $a_n = a_{n/2} + d$, which solves to $a_n = d\log_2(n) + A$, where A is a constant chosen to fit the initial conditions. We assume here that n is a power of 2, to avoid ceilings and floors.
- If $c = 2$ and $f(n)$ is constant, we have $a_n = 2a_{n/2} + d$, which solves to $a_n = An - d$. Our $3n/2 - 2$ steps to find max and min fits into this case.
- If $c = 2$ and $f(n) = dn$, we have $a_n = 2a_{n/2} + dn$, which solves to $a_n = dn(\log_2 n + A)$.

Fast Multiplication

- Normally multiplying two n-bit numbers would require $O(n^2)$ bit multiplications.
- By adding some cheaper additions, though, we can do it with fewer multiplications.
- Write the numbers w_1 and w_2 as u_1v_1 and u_2v_2, where the u's and v's are n/2 bit numbers. Then $w_1 \times w_2 = (u_1 \times u_2)2^n + [(u_1 \times v_2) + (v_1 \times u_2)]2^{n/2} + (v_1 \times v_2)$. We have four products of n/2-bit numbers.

Fast Multiplication

- $(u_1 \times u_2)2^n + [(u_1 \times v_2) + (v_1 \times u_2)]2^{n/2} + (v_1 \times v_2)$ has four products of n/2-bit numbers.
- But if we compute $u_1 \times u_2$, $v_1 \times v_2$, and $(u_1 + v_1) \times (u_2 + v_2)$, using only three multiplications, we can get all three terms we need by addition.
- Our number of multiplications satisfies the recurrence $a_n = 3a_{n/2}$, which turns out to solve to $a_n = n^{\log 3} = n^{1.585\ldots}$, much better than n^2. Of course there are complications like the time for the additions.

The CLRS Master Theorem

- In COMPSCI 311 we learn a theorem called the Master Theorem in the popular CLRS textbook. It gives a solution to the recurrence $a_n = ca_{n/k} + f(n)$, which applies when we divide the size-n problem into c problems of size n/k each, with f(n) overhead to split the problems and merge the solutions.
- The solutions are given in big-O form, befitting a course where we usually regard resource bounds this way.

The CLRS Master Theorem

- We have $a_n = ca_{n/k} + f(n)$.
- The result depends on the relationship between $f(n)$ and $g(n) = n^{\log c}$, where the log is base k. The statement below is approximate.
- If $f(n) = o(g(n))$, then $a_n = \Theta(g(n))$.
- If $f(n) = \Theta(g(n))$, then $a_n = \Theta(g(n)\log n)$.
- If $f(n) = \omega(g(n))$, then $a_n = \Theta(f(n))$.