CS 361, Lecture 9

Jared Saia
University of New Mexico

Outline

- Recurrence Relations
- Recursion Tree Method
- In Class Exercise
- Intro to Annihilators

Recurrence Relations

- $T(n) = 2 * T(n/2) + n$ is an example of a *recurrence relation*
- A *Recurrence Relation* is any equation for a function T, where T appears on both the left and right sides of the equation.
- We always want to "solve" these recurrence relation by getting an equation for T, where T appears on just the left side of the equation

A Note

- In most cases, $T(n) = O(1)$, so we will leave out the "base cases" for recurrences when we want just an asymptotic solution.

Review

- Up to this point, I've been supplying you with good "guesses" for recurrence solutions
- Q: How do we get these guesses?

Getting Good Guesses (I)

Following are some good guesses for solutions to recurrences.
$\log n$
$\sqrt{n}$
n
$n \log n$
n^2
n^3
2^n

Better Techniques (II)

We will review two new techniques:

- Recursion tree method
- Characteristic polynomials

(note: we will *not* cover the Master Theorem given in the book since the method of annihilators is more powerful)

Recursion-tree method

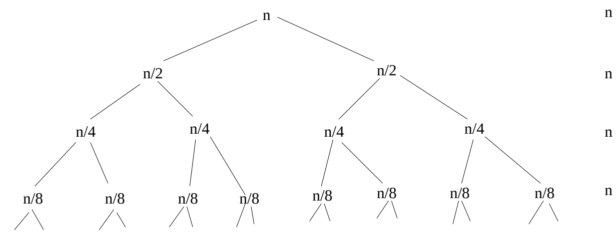
- Each node represents the cost of a single subproblem in a recursive call
- First, we sum the costs of the nodes in each level of the tree
- Then, we sum the costs of all of the levels

Recursion-tree method

- Used to get a good guess which is then refined and verified using substitution method
- Best method (usually) for recurrences where a term like $T(n/c)$ appears on the right hand side of the equality

Example 1

- Consider the recurrence for the running time of Mergesort: $T(n) = 2T(n/2) + n$, $T(1) = O(1)$



Example 1

- We can see that each level of the tree sums to n
- Further the depth of the tree is $\log n$ ($n/2^d = 1$ implies that $d = \log n]$)
- So we can guess that $T(n) = O(n \log n)$

Now Verify!

- We've got a "guess" that $T(n) = O(n \log n)$
- We need to *verify* that this guess is in fact correct
- We verify using induction
- In particular, want to verify that $T(n) \leq cn \log n$ for all $n > 1$

Induction

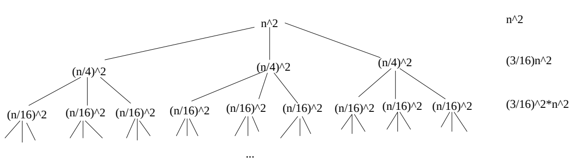
- To show: $T(n) \le cn \log n$ for some constants c, for $n > 1$
- Base Case: $T(2) = O(1)$ by definition. This means $T(2) < k$ for some constant k. Thus we can chose c large enough so that $T(2) < k \le c * 2 \log 2$ is true
- Inductive Hypothesis: For all $j < n$, $T(j) \le cj \log j$
- Inductive step

$$
\begin{aligned}
T(n) &= 2T(n/2) + n && (1)\\
&\le 2(cn/2 \log(n/2)) + n && (2)\\
&= cn \log(n/2) + n && (3)\\
&= cn \log n - cn + n && (4)\\
&= cn \log n && (5)\\
& && (6)
\end{aligned}
$$

Where the last step holds provided that $c > 1$

Example 2

- Let's solve the recurrence $T(n) = 3T(n/4) + n^2$



Example 2

- We can see that the i-th level of the tree sums to $(3/16)^i n^2$.
- Further the depth of the tree is $\log_4 n$
- So we can guess that $T(n) = \sum_{i=0}^{\log_4 n - 1} (3/16)^i n^2$

A guess

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_4 n - 1} (3/16)^i n^2 && (7)\\
&< n^2 \sum_{i=0}^{\infty} (3/16)^i && (8)\\
&= \frac{1}{1 - (3/16)} n^2 && (9)\\
&= O(n^2) && (10)
\end{aligned}
$$

Now Verify!

- We've got a "guess" that $T(n) = O(n^2)$
- We need to *verify* that this guess is in fact correct
- We verify using induction
- In particular, want to verify that $T(n) \le cn^2$, for some constant c.

Induction

- To show: $T(n) \le cn^2$, for some constant c
- Base Case: $T(1) = O(1)$ by definition. This means $T(1) < k$ for some constant k. Thus we can chose c large enough so that $T(1) < k \le c1^2$ is true
- Inductive Hypothesis: For all $j < n$, $T(j) \le cj^2$
- Inductive step

$$
\begin{aligned}
T(n) &= 3T(n/4) + n^2 && (11)\\
&\le 3(c(n/4)^2) + n^2 && (12)\\
&= c(3/16)n^2 + n^2 && (13)\\
&= (c(3/16) + 1)n^2 && (14)\\
&\le cn^2 && (15)\\
& && (16)
\end{aligned}
$$

Where the last step holds provided that $c(3/16) + 1 \le c$, which is true when $c \ge 16/13$

In Class Exercise (I)

Use the recursion tree method to guess a solution to the recursion $T(n) = 2T(n/2) + n^2$. Give the guess in terms of big-O notation:

- Q1: What is the total cost of the 0-th, 1-st and 2-nd level of the tree?
- Q2: What is the total cost of the i-th level of the tree for general i?
- Q3: How many levels of the tree are there?
- Q4: What is the summation giving the total cost of the tree?
- Q5: Give a good upperbound on this summation.

In Class Exercise (II)

Now prove that this guess works using induction!

- Q1: What is the base case? Prove that it holds.
- Q2: What is the inductive hypothesis?
- Q3: What is the inductive step?

Take Away

- Recursion tree method is good for getting "guesses" for recurrences where a term like $T(n/c)$ appears on the right side of the equality
- Once we get the guess, then need to verify using the substitution method
- Recursion trees are useful but limited (they can't help us get guesses for recurrences like $f(n) = f(n-1) + f(n-2)$)

Another Tool

- We'll learn another more powerful method for solving recurrences called *annihilators*
- This will take three to four classes to go over
- Annihilators are similar to "generating functions"

Intro to Annihilators

- Suppose we are given a sequence of numbers $A = \langle a_0, a_1, a_2, \cdots \rangle$
- This might be a sequence like the Fibonacci numbers
- I.e. $A = \langle a_0, a_1, a_2, \ldots) = (T(1), T(2), T(3), \cdots \rangle$

Annihilator Operators

We define three basic operations we can perform on this sequence:

1. Multiply the sequence by a constant: $cA = \langle ca_0, ca_1, ca_2, \cdots \rangle$
2. Shift the sequence to the left: $\mathbf{L}A = \langle a_1, a_2, a_3, \cdots \rangle$
3. Add two sequences: if $A = \langle a_0, a_1, a_2, \cdots \rangle$ and $B = \langle b_0, b_1, b_2, \cdots \rangle$, then $A + B = \langle a_0 + b_0, a_1 + b_1, a_2 + b_2, \cdots \rangle$

Annihilator Description

- We first express our recurrence as a sequence T
- *We use these three operators to "annihilate" T, i.e. make it all 0's*
- Key rule: can't multiply by the constant 0
- We can then determine the solution to the recurrence from the sequence of operations performed to annihilate T

Example

- Consider the recurrence $T(n) = 2T(n-1)$, $T(0) = 1$
- If we solve for the first few terms of this sequence, we can see they are $\langle 2^0, 2^1, 2^2, 2^3, \cdots \rangle$
- Thus this recurrence becomes the sequence:

$$T = \langle 2^0, 2^1, 2^2, 2^3, \cdots \rangle$$

Example (II)

Let's annihilate $T = \langle 2^0, 2^1, 2^2, 2^3, \cdots \rangle$

- Multiplying by a constant $c = 2$ gets:

$$2T = \langle 2 * 2^0, 2 * 2^1, 2 * 2^2, 2 * 2^3, \cdots \rangle = \langle 2^1, 2^2, 2^3, 2^4, \cdots \rangle$$

- Shifting one place to the left gets $\mathbf{L}T = \langle 2^1, 2^2, 2^3, 2^4, \cdots \rangle$
- Adding the sequence $\mathbf{L}T$ and $-2T$ gives:

$$\mathbf{L}T - 2T = \langle 2^1 - 2^1, 2^2 - 2^2, 2^3 - 2^3, \cdots \rangle = \langle 0, 0, 0, \cdots \rangle$$

- The annihilator of T is thus $\mathbf{L} - 2$

Todo

- Start hw2!