Lecture 2: Mathematical Prelimaries

Read: Chapt 1 of Weiss and skim Chapt 2.

Mathematics: Although this course will not be a "theory" course, it is important to have a basic understanding of the mathematical tools that will be needed to reason about the data structures and algorithms we will be working with. A good understanding of mathematics helps greatly in the ability to design good data structures, since through mathematics it is possible to get a clearer understanding of the nature of the data structures, and a general feeling for their efficiency in time and space. Last time we gave a brief introduction to asymptotic (big-"Oh" notation), and later this semester we will see how to apply that. Today we consider a few other preliminary notions: summations and proofs by induction.

Summations: Summations are important in the analysis of programs that operate iteratively. For example, in the following code fragment

```
for (i = 0; i < n; i++) { ...  }
```

Where the loop body (the "...") takes $f(i)$ time to run the total running time is given by the summation

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

Observe that nested loops naturally lead to nested sums. Solving summations breaks down into two basic steps. First simplify the summation as much as possible by removing constant terms (note that a constant here means anything that is independent of the loop variable, i) and separating individual terms into separate summations. Then each of the remaining simplified sums can be solved. Some important sums to know are

$$\begin{aligned}
\sum_{i=1}^{n} 1 &= n && \text{(The constant series)}\\
\sum_{i=1}^{n} i &= \frac{n(n+1)}{2} && \text{(The arithmetic series)}\\
\sum_{i=1}^{n} \frac{1}{i} &= \ln n + O(1) && \text{(The harmonic series)}\\
\sum_{i=0}^{n} c^i &= \frac{c^{n+1}-1}{c-1} \qquad c \neq 1 && \text{(The geometric series)}
\end{aligned}$$

Note that complex sums can often be broken down into simpler terms, which can then be solved. For example

$$\begin{aligned}
T(n) &= \sum_{i=n}^{2n-1} (3+4i) = \sum_{i=0}^{2n-1} (3+4i) - \sum_{i=0}^{n-1} (3+4i)\\
&= \left(3\sum_{i=0}^{2n-1} 1 + 4\sum_{i=0}^{2n-1} i\right) - \left(3\sum_{i=0}^{n-1} 1 + 4\sum_{i=0}^{n-1} i\right)\\
&= \left(3(2n) + 4\frac{2n(2n-1)}{2}\right) - \left(3(n) + 4\frac{n(n-1)}{2}\right) = (n + 6n^2).
\end{aligned}$$

The last summation is probably the most important one for data structures. For example, suppose you want to know how many nodes are in a complete 3-ary tree of height h. (We have not given a formal definition of tree's yet, but consider the figure below.

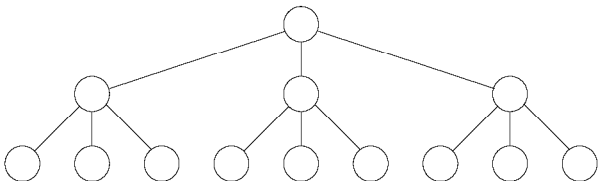

Figure 1: Complete 3-ary tree of height 2.

The *height* of a tree is the maximum number of edges from the root to a leaf.) One way to break this computation down is to look at the tree level by level. At the top level (level 0) there is 1 node, at level 1 there are 3 nodes, at level 2, 9 nodes, and in general at level i there 3^i nodes. To find the total number of nodes we sum over all levels, 0 through h. Plugging into the above equation with $h = n$ we have:

$$\sum_{i=0}^{h} 3^i = \frac{3^{h+1} - 1}{2} \in O(3^h).$$

Conversely, if someone told you that he had a 3-ary tree with n nodes, you could determine the height by inverting this. Since $n = (3^{(h+1)} - 1)/2$ then we have

$$3^{(h+1)} = (2n + 1)$$

implying that

$$h = (\log_3(2n + 1)) - 1 \in O(\log n).$$

Another important fact to keep in mind about summations is that they can be approximated using integrals.

$$\sum_{i=a}^{b} f(i) \approx \int_{x=a}^{b} f(x)dx.$$

Given an obscure summation, it is often possible to find it in a book on integrals, and use the formula to approximate the sum.

Recurrences: A second mathematical construct that arises when studying recursive programs (as are many described in this class) is that of a recurrence. A recurrence is a mathematical formula that is defined recursively. For example, let's go back to our example of a 3-ary tree of height h. There is another way to describe the number of nodes in a complete 3-ary tree. If $h = 0$ then the tree consists of a single node. Otherwise that the tree consists of a root node and 3 copies of a 3-ary tree of height $h - 1$. This suggests the following recurrence which defines the number of nodes $N(h)$ in a 3-ary tree of height h:

$$\begin{aligned} N(0) &= 1 \\ N(h) &= 3N(h-1) + 1 \qquad \text{if } h \geq 1. \end{aligned}$$

Although the definition appears circular, it is well grounded since we eventually reduce to $N(0)$.

$$\begin{aligned} N(1) &= 3N(0) + 1 = 3 \cdot 1 + 1 = 4 \\ N(2) &= 3N(1) + 1 = 3 \cdot 4 + 1 = 13 \\ N(3) &= 3N(2) + 1 = 3 \cdot 13 + 1 = 40, \end{aligned}$$

and so on.

There are two common methods for solving recurrences. One (which works well for simple regular recurrences) is to repeatedly expand the recurrence definition, eventually reducing it to a summation, and the other is to just guess an answer and use induction. Here is an example of the former technique.

$$\begin{aligned} N(h) &= 3N(h-1)+1 \\ &= 3(3N(h-2)+1)+1 \;=\; 9N(h-2)+3+1 \\ &= 9(3N(h-3)+1)+3+1 \;=\; 27N(h-3)+9+3+1 \\ &\vdots \\ &= 3^k N(h-k) + (3^{k-1}+\ldots+9+3+1) \end{aligned}$$

When does this all end? We know that $N(0) = 1$, so let's set $k = h$ implying that

$$N(h) = 3^h N(0) + (3^{h-1}+\ldots+3+1) = 3^h + 3^{h-1} + \ldots + 3 + 1 = \sum_{i=0}^{h} 3^i.$$

This is the same thing we saw before, just derived in a different way.

Proofs by Induction: The last mathematical technique of importance is that of proofs by induction. Induction proofs are critical to all aspects of computer science and data structures, not just efficiency proofs. In particular, virtually all correctness arguments are based on induction. From courses on discrete mathematics you have probably learned about the standard approach to induction. You have some theorem that you want to prove that is of the form, "For all integers $n \geq 1$, blah, blah, blah", where the statement of the theorem involves n in some way. The idea is to prove the theorem for some basis set of n-values (e.g. $n = 1$ in this case), and then show that if the theorem holds when you plug in a specific value $n-1$ into the theorem then it holds when you plug in n itself. (You may be more familiar with going from n to $n+1$ but obviously the two are equivalent.)

In data structures, and especially when dealing with trees, this type of induction is not particularly helpful. Instead a slight variant called *strong induction* seems to be more relevant. The idea is to assume that if the theorem holds for all values of n that are strictly less than n then it is true for n. As the semester goes on we will see examples of strong induction proofs.

Let's go back to our previous example problem. Suppose we want to prove the following theorem.

Theorem: Let T be a complete 3-ary tree with $n \geq 1$ nodes. Let $H(n)$ denote the height of this tree. Then

$$H(n) = (\log_3(2n+1)) - 1.$$

Basis Case: (Take the smallest legal value of n, $n = 1$ in this case.) A tree with a single node has height 0, so $H(1) = 0$. Plugging $n = 1$ into the formula gives $(\log_3(2 \cdot 1 + 1)) - 1$ which is equal to $(\log_3 3) - 1$ or 0, as desired.

Induction Step: We want to prove the theorem for the specific value $n > 1$. Note that we cannot apply standard induction here, because there is no complete 3-ary tree with 2 nodes in it (the next larger one has 4 nodes).

We will assume the induction hypothesis, that for all smaller n', $1 \leq n' < n$, $H(n')$ is given by the formula above. (This is sometimes called *strong induction*, and it is good to learn since most induction proofs in data structures work this way.)

Let's consider a complete 3-ary tree with $n > 1$ nodes. Since $n > 1$, it must consist of a root node plus 3 identical subtrees, each being a complete 3-ary tree of $n' < n$ nodes. How many nodes are in these subtrees? Since they are identical, if we exclude the root node, each subtree has one third of the remaining number nodes, so $n' = (n-1)/3$. Since $n' < n$ we can apply the induction hypothesis. This tells us that

$$\begin{aligned} H(n') &= (\log_3(2n'+1)) - 1 = (\log_3(2(n-1)/3+1)) - 1 \\ &= (\log_3(2(n-1)+3)/3) - 1 = (\log_3(2n+1)/3) - 1 \\ &= \log_3(2n+1) - \log_3 3 - 1 = \log_3(2n+1) - 2. \end{aligned}$$

Note that the height of the entire tree is one more than the heights of the subtrees so $H(n) = H(n') + 1$. Thus we have:

$$H(n) = \log_3(2n+1) - 2 + 1 = \log_3(2n+1) - 1,$$

as desired.

This may seem like an awfully long-winded way of proving such a simple fact. But induction is a very powerful technique for proving many more complex facts that arise in data structure analyses.

You need to be careful when attempting proofs by induction that involve $O(n)$ notation. Here is an example of a common error.

False! Theorem: For $n \geq 1$, let $T(n)$ be given by the following summation

$$T(n) = \sum_{i=0}^{n} i,$$

then $T(n) \in O(n)$. (We know from the formula for the linear series above that $T(n) = n(n+1)/2 \in O(n^2)$. So this must be false. Can you spot the error in the following "proof"?)

Basis Case: For $n = 1$ we have $T(1) = 1$, and 1 is $O(1)$.

Induction Step: We want to prove the theorem for the specific value $n > 1$. Suppose that for any $n' < n$, $T(n') \in O(n')$. Now, for the case n, we have (by definition)

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

Now, since $n-1 < n$, we can apply the induction hypothesis, giving $T(n-1) \in O(n-1)$. Plugging this back in we have

$$T(n) \in O(n-1) + n.$$

But $(n-1) + n \leq 2n - 1 \in O(n)$, so we have $T(n) \in O(n)$.

What is the error? Recall asymptotic notation applies to arbitrarily large n (for n in the limit). However induction proofs by their very nature only apply to specific values of n. The proper way to prove this by induction would be to come up with a concrete expression, which does not involve O-notation. For example, try to prove that for all $n \geq 1$, $T(n) \leq 50n$. If you attempt to prove this by induction (try it!) you will see that it fails.