# Trees and Structural Induction

Margaret M. Fleck

2 April 2010

This lecture covers some basic properties of trees (section 10.1 of Rosen and introduces structural induction (in section 4.3 of Rosen).

## 1 Announcements

Midterm coming up next Wednesday. Contact me if you have a conflict. Also, folks doing the honors add-on: second homework is due next Monday (5th).

## 2 Counting nodes

Recall that a *full m-ary tree* is a tree in which each node has either m children or no children. So, in a full binary tree, each node has two or zero children. Remember also that internal nodes are nodes with children and leaf nodes are nodes without children.

Claim: A full m-ary tree with $i$ internal nodes has $mi + 1$ nodes total.

To see why this is true, notice that there are two types of nodes: nodes with a parent and nodes without a parent. A tree has exactly one node with no parent. We can count the nodes with a parent by taking the number of parents in the tree ($i$) and multiplying by the branching factor $m$.

Therefore, the number of leaves in a full m-ary tree with $i$ internal nodes is $(mi + 1) - i = (m - 1)i + 1$.

Rosen lists several more similar equations relating number of nodes, number of leaves, and number of internal nodes.

## 3 Height vs number of nodes

Recall that the level of a node is the number of edges in the path from it to the root. That is, the root has level 0. The height of a tree is the maximum level of any (leaf) node.

Now, suppose that we have a binary tree of height $h$. How many nodes and how many leaves does it contain? This clearly can't be an exact formula, since some trees are more bushy than others and some are more balanced than others (all leaves at approximately the same level). But we can give useful upper and lower bounds.

To minimize the node counts, consider a tree that has just one leaf. It contains $h + 1$ nodes connected into a straight line by $h$ edges. So the minimum number of leaves is 1 (regardless of $h$) and the minimum number of nodes is $h + 1$.

The node counts are maximized by a tree which is full (see above) and complete (all leaves are at the same level). In that case, the number of leaves is $2^h$ and the number of nodes is $\sum_{L=0}^{h} 2^L = 2^{h+1} - 1$.

So for a full, complete binary tree, the total number of nodes $n$ is $\Theta(2^h)$. So then $h$ is $\Theta(\log_2 n)$. If the tree might not be full and complete, this is a lower bound on the height, so $h$ is $\Omega(\log_2 n)$. The same applies to the number of leaves.

In a "balanced" m-ary tree of height $h$, all leaves are either at height $h$ or at height $h-1$. Balanced trees are useful when you want to store $n$ items (where $n$ is some random natural number that might not be a power of 2) while keeping all the leaves at approximately the same height. Balanced trees aren't as rigid as full binary trees, but they also have $\Theta(\log_2 n)$ height. This means that all the leaves are fairly close to the root, which leads to good behavior from algorithms trying to store and find things in the tree.

## 4 Tree induction

We claimed that

**Claim 1** *Let $T$ be a binary tree, with height $h$ and $n$ nodes. Then $n \leq 2^{h+1} - 1$.*

We can prove this claim by induction. Our induction variable needs to be some measure of the size of the tree, e.g. its height or the number of nodes in it. Whichever variable we choose, it's important that the inductive step divide up the tree at the top, into a root plus (for a binary tree) two subtrees.

> Proof by induction on $h$, where $h$ is the height of the tree.
>
> Base: The base case is a tree consisting of a single node with no edges. It has $h = 0$ and $n = 1$. Then we work out that $2^{h+1} - 1 = 2^1 - 1 = 1 = n$.
>
> Induction: Suppose that the claim is true for all binary trees of height $< h$, where $h > 0$. Let $T$ be a binary tree of height $h$.
>
> Case 1: $T$ consists of a root plus one subtree $X$. $X$ has height $h-1$. So $X$ contains at most $2^h - 1$ nodes. And then $X$ contains at most $2^h$ nodes, which is less than $2^{h+1} - 1$.
>
> Case 2: $T$ consists of a root plus two subtrees $X$ and $Y$. $X$ and $Y$ have heights $p$ and $q$, both of which have to be less than $h$, i.e. $\leq h - 1$. $X$ contains at most $2^{p+1} - 1$ nodes and $Y$ contains at most $2^{q+1} - 1$ nodes, by the inductive hypothesis. But this means that $X$ and $Y$ each contain $\leq 2^h - l$ nodes.
>
> So the total number of nodes in $T$ is the number of nodes in $X$ plus the number of nodes in $Y$ plus one (the new root node). This is $\leq 1 + (2^p - 1) + (2^q - 1) \leq 1 + 2(2^h - 1) = 1 + 2^{h+1} - 2 = 2^{h+1} - 1$
>
> So the total number of nodes in $T$ is $\leq 2^{h+1} - 1$, which is what we needed to show. $\square$

## 5 Structural induction

Inductive proofs on trees can also be written using "structural induction." In structural induction, there is no explicit induction variable. Rather, the outline of the proof follows the structure of a recursive definition. This is slightly simpler for trees and a lot simpler for some other types of examples (e.g. we'll see one on Monday) where a suitable induction variable can be hard to find.

To do structural induction on trees, we need a recursive definition of trees. To keep this simple, we'll stick to full binary trees. A full binary tree can be defined recursively as follows:

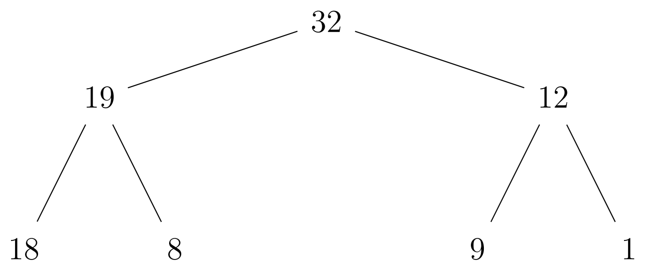
- A single node is a full binary tree (its root).
- Suppose $X$ and $Y$ are full binary trees. Define a new tree $T$ to be the tree which consists of a (new) root node $x$ to which the root nodes of $X$ and $Y$ are attached as children. Then $T$ is also a full binary tree.

If we wanted to define any binary tree, including those that aren't full, we'd need to add a second recursive clause to our definition:

- A single node is a binary tree (its root).
- Suppose $X$ and $Y$ are binary trees. Define a new tree $T$ to be the tree which consists of a (new) root node $x$ to which the root nodes of $X$ and $Y$ are attached as children. Then $T$ is also a binary tree.
- Suppose $X$ is a binary tree, then so the new tree $T$ which consists of a root node with the root of $X$ as its (single) child.

## 6 Heap example

To have a nice claim to prove by structural induction, suppose we store numbers in the nodes of a full binary tree. The numbers obey the *heap property* if, for every node $X$ in the tree, the value in $X$ is at least as big as the value in each of $X$'s children. For example:



Notice that the values at one level aren't uniformly bigger than the values at the next lower level. For example, 18 in the bottom level is larger than

12 on the middle level. But values never decrease as you move along a path from a leaf up to the root.

Trees with the heap property are convenient for applications where you have to maintain a list of people or tasks with associated priorities. It's easy to retrieve the person or task with top priority: it lives in the root. And it's easy to restore the heap property if you add or remove a person or task.

I claim that:

**Claim 2** *If a tree has the heap property, then the value in the root of the tree is at least as large as the value in any node of the tree.*

To keep the proof simple, let's restrict our attention to full binary trees:

**Claim 3** *If a full binary tree has the heap property, then the value in the root of the tree is at least as large as the value in any node of the tree.*

Let's let $v(X)$ be the value at node $X$ and let's use the recursive structure of trees to do our proof.

> Proof by structural induction.
>
> Base: If a tree contains only one node, obviously the largest value in the tree lives in the root!
>
> Induction: Suppose that the claim is true for trees $X$ and $Y$. We need to show that the claim is also true for the tree $T$ that consists of a root node plus subtrees $X$ and $Y$.
>
> Let $r$ be the root of the whole tree $T$. Suppose $p$ and $q$ are the children of $r$, i.e. the root nodes of $X$ and $Y$. Since $T$ has the heap property, $v(r) \geq v(p)$ and $v(r) \geq v(q)$.
>
> Suppose that $x$ is any node of $T$. We need to show that $v(r) \geq v(x)$. There are three cases:
>
> Case 1: $x = r$. This is obvious.
>
> Case 2: $x$ is any node in the subtree $X$. By the inductive hypothesis $v(p) \geq v(x)$. But we know that $v(r) \geq v(p)$. So $v(p) \geq v(x)$.
>
> Case 3: $x$ is any node in the subtree $Y$. Similar to case 2.
>
> So, for any node $x$ in $T$, $v(r) \geq v(x)$.$\square$

In the inductive step, notice that we split up the big tree ($T$) at its root, producing two smaller subtrees ($X$) and ($Y$). Some students try to do induction on trees by grafting stuff onto the bottom of the tree. This frequently does not work, especially as you get to examples in more advanced courses. Therefore, we will take off points if you do it on homework or tests.