Maggie Johnson
CS109

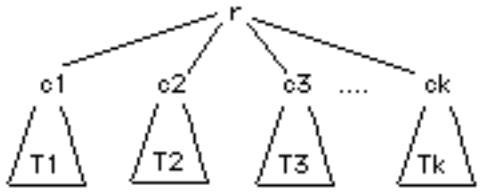
Trees II: Structural Induction & Proofs on Trees

Key topics:

* Structural Induction
* Proving the Correctness of Recursive Functions that Act on Trees
* Other Methods of Proof for Trees
* A Classic Tree (Node) Induction

Earlier this quarter, you saw a number of inductive proofs concerning integers and sequences. We would assume some statement for n, or for all integers less than or equal to n (strong induction), and use this "inductive hypothesis" to prove the same statement for n+1. I'm sure you will be delighted to learn that a similar form of proof is useful for concluding facts about trees.

Suppose we want to prove that a statement S(T) is true for all trees. For a basis, we show that S(T) is true whenever T consists of one node, i.e. T is a single internal node. For the induction, we suppose that T is a tree with root r and children c1, c2, ..., ck. Let T1, T2, ..., Tk be the subtrees of T rooted at c1, c2, ..., ck respectively as shown below:



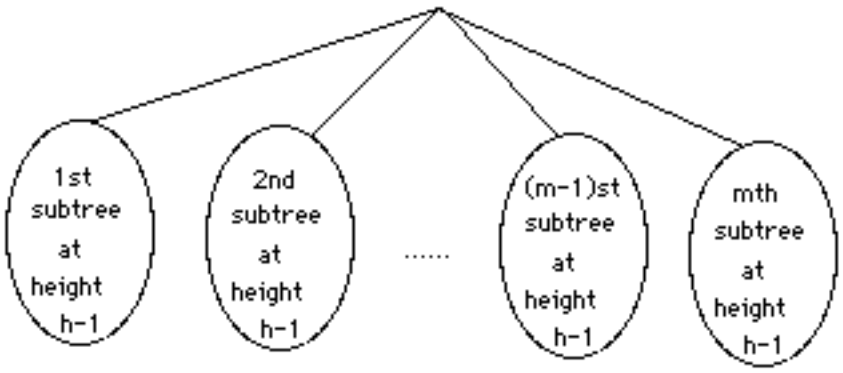
The inductive step is to assume S(T1), S(T2), ..., S(Tk) are all true, and prove S(T). If we can do this, we have proven S(T).

Why does structural induction work?

Example 1: S(T) denotes that there are at most m^h leaves in an m-ary tree of height h. (Note: the root is at level 1 with a height of 0)

> basis: Consider an m-ary tree of height 1. These trees consist of a root with no more than m children, each of which is a leaf. Therefore, there are no more than $m^1 = m$ leaves in an m-ary tree of height 1.
> inductive hypothesis: We assume the above statement is true for all m-ary trees of height less than h, and show it is true for all m-ary trees of height h.

PROOF:
Let T be an m-ary tree of height h. The leaves of T are the leaves of the subtrees of T obtained by deleting the edges from the root to each of the vertices connected to it, as shown below:



Each of these subtrees has height less than or equal to h-1. So, by the inductive hypothesis, these trees have at most $m^{(h-1)}$ leaves. Since there are at most m such subtrees, each with a minimum of $m^{(h-1)}$ leaves, there are at most $m * m^{(h-1)} = m^h$ leaves in the rooted tree.

Thus, by the principle of mathematical induction applied to the structure of trees, S(T) is true for all m-ary trees.

Example 2: S(T) denotes a *strictly* binary tree with n leaves has 2n-2 edges.

basis: A strictly binary tree with two leaves consists of the root connected to its only two children: 2 * 2 - 2 = 2.
inductive hypothesis: We assume S(T) is true for all strictly binary trees with i leaves where i >= 2 and i < n leaves; Show that S(T) is true for a tree with n leaves.

PROOF:
Consider a tree T with n leaves where n > 1. By the definition of a binary tree, T must consist of two subtrees T_L and T_R (since n > 1). If T_L has n_L leaves and T_R has n_R leaves, then $n_L + n_R = n$. By the inductive hypothesis, T_L has $2n_L - 2$ edges and T_R has $2n_R - 2$ edges. Since T has two additional edges that are not part of T_L or T_R, the number of edges in T is $(2n_L - 2) + (2n_L - 2) + 2 = 2(n_L + n_R) - 2 = 2n - 2$.

Thus, by the principle of mathematical induction applied to the structure of trees, S(T) is true for all strictly binary trees.

How does structural induction differ from induction on integers?

Proving the Correctness of Recursive Functions that Act on Trees

A structural induction is generally needed to prove the correctness of a recursive program that acts on trees. For example, eval from the last handout:

```
int eval(tree pT)
{
        struct tnode *secondchild;

1       if (pT->op == 'i')
2               return (pT->value);
        else {
3               pT->value = eval(pT->lmost);
4               secondchild = pT->lmost->rsibling;

5               switch (pT->op) {
6                       case '+': pT->value = pT->value + eval(secondchild);
                                break;
7                       case '-': pT->value = pT->value - eval(secondchild);
                                break;
8                       case '*': pT->value = pT->value * eval(secondchild);
                                break;
9                       case '/': pT->value = pT->value / eval(secondchild);
                                break;
10                      }
                return  pT->value;
        }
}
```

We will prove by structural induction the following statement S(T):

The value returned by eval when called on the root of T equals the value of the arithmetic expression represented by T.

basis: T consists of a single node. When eval(pT) is called, pT's op field = 'i' and eval returns the value at line 2.
inductive hypothesis: Suppose T is not a single node. The inductive hypothesis is that S(T') is true for each tree T' rooted at one of the children of T. We must use these facts to prove S(T) for the tree rooted at T.

PROOF:
Since our operators are binary, we assume that T has two children. Let the values of the expressions rooted at these two subtrees be v1 and v2, respectively.

First, we use the inductive hypothesis to conclude that the value computed at line 3 and stored in pT->value is v1; eval is called on the first child of pT on line 3 so by the inductive hypothesis, the value returned by the call is the value of the expression rooted at that tree (v1). We also can conclude by the inductive hypothesis that the call to eval(secondchild) in the switch statement (whichever case applies) will produce v2.

We know that pT->value = v1 when we get to the case statement, so the assignments in the case statement assign to pT->value (v1 op v2) with op being whatever operator is at pT. In any of the four cases, the value at line 10 which is

returned is (v1 op v2), i.e., the value of the expression rooted at T. This is what we wanted to prove.

Thus, by the principle of mathematical induction applied to the structure of trees, S(T) is true.

Recall from last time, the following incorrect function:

```
        Boolean IsBST(tree pT)
        {
                Boolean LeftIsBST, RightIsBST;

1               if  (pT == NULL)
2                       return TRUE;
                else {
3                       if (pT->left == NULL)
4                               LeftIsBST = TRUE;
                        else
5                               if ((strcmp(pT->left->word, pT->word)) > 0)

6                                       LeftIsBST = FALSE;
                                else
7                                       LeftIsBST = IsBST(pT->left);

8                       if (pT->right == NULL)
9                               RightIsBST = TRUE;
                        else
10                              if ((strcmp(pT->right->word, pT->word)) < 0)
11                                      RightIsBST = FALSE;
                                else
12                                      RightIsBST = IsBST(pT->right);

13                      return ((LeftIsBST) && (RightIsBST));
                }
        }
```

Let's see what happens when we do a structural inductive proof to show correctness:

S(T) denotes: The boolean value returned by IsBST when called on the root of T is TRUE if T is a binary search tree and FALSE otherwise.

basis: T consists of a single node. When IsBST(pT) is called, pT->left is NULL, so line 3 is TRUE and LeftIsBST is set to TRUE on line 4. Similarly, line 8 is TRUE and RightIsBST is set to TRUE on line 9. Finally, line 13 returns TRUE && TRUE.
inductive hypothesis: Suppose T is not a single node. The inductive hypothesis is that S(T') is true for each the T' rooted at one of the children of T. We must use these facts to prove S(T) for the tree rooted at T.

PROOF:
……

What is wrong with this proof?

Other Methods of Proof for Trees

There are other ways of proving properties about trees, by just using some logic, some knowledge about the structure of trees, and a little common sense (or luck). For example:

Example 4: Prove that a full m-ary tree with i internal nodes contains mi + 1 nodes.

> We need the following definitions: A full m-ary tree is one where every node in the tree (except the leaves which are all at the same level) has m children. An internal node is any node that is not a leaf.
>
> PROOF:
> Every node, except the root, is the child of an internal node. Since this tree is full, each of these i internal nodes has m children, so there are mi nodes in the tree other than the root. Include the root and you get mi + 1 nodes.

Other useful theorems (left to the reader to prove for fun and practice):

1) A tree with n vertices has n-1 edges (a classic node induction, see below)
2) A full m-ary tree with
   a) n vertices has i = (n-1)/m internal vertices and L = [(m-1)n+1]/m leaves.
   b) i internal vertices has n = mi + 1 vertices and L = (m-1)i + 1 leaves.
   c) L leaves has n = (mL - 1)/(m - 1) vertices and i = (L - 1)/(m - 1) internal vertices.

A Classic Tree (Node) Induction

Proof: A tree with *n* nodes has *n*-1 edges

By induction on the number of nodes.

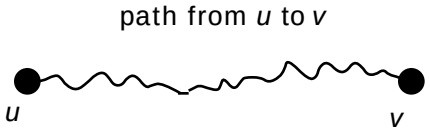
Base case: A tree with 1 node has 0 edges, since loops are not allowed in trees.

Induction:

Prove: A tree with *n*+1 nodes has n edges
Assuming: A tree with *n* nodes has *n*-1 edges.

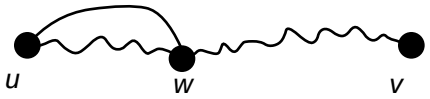
Begin with a tree *T* with *n*+1 nodes. By the lemma below, we can show that this tree has at least one leaf *l*. (We define a leaf as a node of degree 1.) Form a new tree *T'* by removing *l* and the edge incident on *l* from *T*. The tree *T'* now contains *n* nodes, and so by the inductive hyposthesis, T' has n-1 edges. (We know that *T'* is a tree, since *T* was a tree and removing a leaf from *T* does not introduce any cycles, nor does it destroy connectivity.) Since we removed only one edge to go from *T* to *T'*, *T* must have *n* edges, which is what we wanted to show.

Lemma: Any tree with more than one node has at least two nodes of degree one. Consider a tree *T*. There must be at least one path of maximal length in *T*. Find this path *p* and say it has endpoints at nodes *u* and *v*.



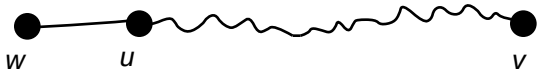
Both *u* and *v* must be of degree one. We can prove this by assuming that one end node, say *u*, has degree 2 (or more). There are two possibilities:

1) There is a node *w* such that a path to *w* can be found starting at either of the two edges of *u*.



But this is impossible; it implies that there is a cycle, and trees are acyclic.

2) One edge leads to *v*, the other does not.



This too is impossible, because in this case *p* is not the longest path in *T*, contradicting our earlier assumption.

To give you a little practice in being very careful about how you do node inductions, the following version of the above proof is incorrect because it goes the wrong way. It begins with a small (*n*) tree and then tries to build a larger (*n*+1) tree.

Incorrect Proof: A tree with *n* nodes has *n*-1 edges

By **faulty** induction on the number of nodes.

Base case: A tree with 1 node has 0 edges, since loops are not allowed in trees.

Induction:

Prove: A tree with *n*+1 nodes has *n* edges

Assuming: A tree with *n* nodes has *n*-1 edges.

(So far, everything is OK...)

Begin with a tree *T* with *n* nodes. (***OOPS!***) By the inductive hypothesis, *T* has *n*-1 edges. We can get a tree with *n*+1 nodes from *T* by adding a node of degree 1. Thus to get a tree *T*' with *n*+1 nodes, we add a single edge and a single node to *T*. Thus *T*' will have *n* edges.

What's wrong? It's the statement "We can get a tree with *n*+1 nodes from *T* by adding a node of degree 1." This is certainly true, but it is dangerous. The danger comes because it may not be the *only* way to get a tree with *n*+1 nodes. For this proof to be correct, that assertion needs to be both stated and proved. A correct proof does not "push out" from n

to n+1 nodes because we do not reach all possible trees that way. Rather we must "lean back" by starting with an (n+1)-node tree and carefully selecting a node to remove to get to an n-node tree.

Why be so picky about something that is "obviously true?" As we get into graph theory, we will see why these little distinctions can cause us to make incorrect assumptions.

Bibliography, and Additional Information

Much of the above material on structural induction was adapted from Aho & Ullman. Additional structural induction proofs can be found in Gersting, and Helman. Refer also to Rosen for other proofs on trees.

A. Aho, J.D. Ullman, *Foundations of Computer Science*, New York: W.H. Freeman, 1992.

J. Gersting, *Mathematical Structures for Computer Science, 2nd ed.*, San Francisco: W.H. Freeman, 1992.

P. Helman, R. Veroff, F. Carrano, *Intermediate Problem Solving and Data Structures, Walls and Mirrors*, Redwood City, CA: Benjamin Cummings, 1991.

K. Rosen, *Discrete Mathematics and its Applications 2nd Ed.*, New York: McGraw-Hill, 1991.