Tree Traversal

Ben Gunning

August 24, 2015

Abstract

1 Introduction

Tree traversal is a topic that is particularly useful in computer science. In particular, tree traversal is useful for search algorithms, as it helps to characterize the positions of nodes in a tree. This enables a search algorithm, a binary search in particular, to perform an efficient search.

In order to understand the intuition behind tree traversal, it is first necessary to lay some foundation. The definition of a tree is an ample place to start.

Definition 1.1 (Tree)**.** A tree is a finite set of nodes and 1-directional edges (think rays, arrows pointing from one node to another node that can only be followed in one direction) satisfying several conditions.

1. There does not exist a circuit of nodes. This means that for any node, there is no path of edges that starts from that node and ends at the same node.

2. No two edges are directed at the same node. More formally, no node has multiple parents.

3. The set is connected. In other words, if you allow edges to be traversed both ways, there exists a path between any two nodes.

A couple of definitions make naming and reference to certain nodes and relationships between nodes easier and more intuitive.

Definition 1.2. A node is a *parent* of another node if there is an edge from the former to the latter. The latter node would be said to be a *child* of the former node. The set of all children of the children of ... of a node is the set of that node's *descendents*. Similarly, the set containing a node's parent, the parent's parent, and so on (to the root) is called that node's *ancestors*.

Next we should define a root, an aspect of a tree. We will also need to prove some characteristics such as the existence and uniqueness of roots.

Definition 1.3 (Root)**.** A root is a node in a tree that has no parents. Alternatively, a root is a node to which no edges point.

Theorem 1.4 (Existence of Roots)**.** *Every tree has a root.*

Proof. We proceed with a proof by contradiction. Consider a tree with no root. Since we've defined trees to be necessarily finite, we may label nodes as $1, 2, ..., n$. Start with node n and consider its parent. There are $n-1$ possible parents for node n. Without loss of generality, suppose that $n-1$ is the parent of n. Next consider node $n-1$. It has $n-2$ possible parents, since n cannot be a parent to $n-1$ without violating the first property of trees. Without loss of generality, let $n-2$ be the parent of $n-1$. We may use the procedure, noting that no previously considered nodes could be a parent to a newly considered node without violating the first property, until we have considered all but one nodes. Without loss of generality, let the last node be 1. There are no possible parents for node 1 that do not violate the first property of trees. Hence, 1 has no parents, and we have a contradiction. ∎

Theorem 1.5 (Uniqueness of Roots)**.** *Every tree has no more than one root.*

Proof. Consider a tree of n-many nodes (1 through n). Let nodes 1 and 2 be two roots of our tree. Since 1 and 2 are two nodes, by the third property, there is a path (ignoring orientation) between 1 and 2. We have three possible cases to consider for this path.

- All of the edges in the path from 1 to 2 form a chain (observing orientation) from 1 to 2. This contradicts our claim that 2 is a root.

- All of the edges in the path from 1 to 2 form a chain (observing orientation) from 2 to 1. This contradicts our claim that 1 is a root.

- Two edges in the path from 1 to 2 point to the same node. This contradicts the second property.

These cases are exhaustive, and each leads to a contradiction. Hence, the assumption must be false. ∎

These two theorems prove that every tree has exactly one root (using our aforementioned definition). In contrast, there is no limit on the number of nodes of the next type in a given tree.

Definition 1.6 (Leaf)**.** A leaf is a node that is the parent to no other nodes.

.

By the same logic as our previous existence proof, it is easy to prove the following result.

Theorem 1.7. *Every tree has a leaf.*

With this foundation on trees in mind, we will now introduce a type of tree that is particularly interesting for the development of search algorithms.

Definition 1.8 (Binary Tree)**.** A binary tree is a tree in which every node is the parent of no more than two child nodes.

Binary trees are useful for the purpose of binary search algorithms. However, for these purposes, there is a logical question that follows: How can we characterize a binary tree more simply?

2   Characterization
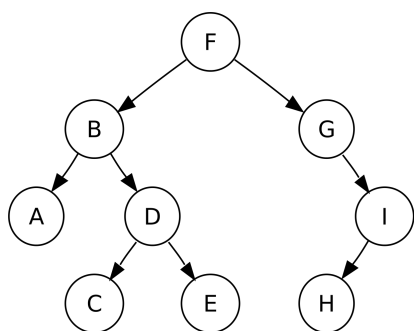
Consider the binary tree above. How can we represent an arbitrary binary tree such as this by means of an ordered list of nodes? In order to do this, we will rely on recursively deconstructing our trees into smaller *subtrees*.

Definition 2.1. Given a node within a binary tree, we may define its children (if any) by the terminology left child and right child, assigned to children based on a general direction that is consistent across all references to the tree. The tree formed by the left child and all of its descendents is called the *left subtree* of the original node. Similarly, the tree formed by the right child and all of its descendents is called the *right subtree* of the original node.

Note that the root, the root's left subtree, and the root's right subtree provide the foundation upon which we may construct a recursive method of ordering the nodes of a binary tree. We shall now define several such methods of ordering the nodes of a binary tree.

Definition 2.2 (Pre-Order Traversal)**.** Given a binary tree, we can perform what is called a *pre-order traversal* of the tree by applying a recursive operation (beginning with the root) in the following manner:

1. Display the node

2. Recursively perform this operation on the node's left subtree

3. Recursively perform this operation on the node's right subtree

Definition 2.3 (In-Order Traversal)**.** Given a binary tree, we can perform what is called an *in-order traversal* of the tree by applying a recursive operation (beginning with the root) in the following manner:

1. Recursively perform this operation on the node's left subtree

2. Display the node

3. Recursively perform this operation on the node's right subtree

Definition 2.4 (Post-Order Traversal)**.** Given a binary tree, we can perform what is called a *post-order traversal* of the tree by applying a recursive operation (beginning with the root) in the following manner:

1. Recursively perform this operation on the node's left subtree

2. Recursively perform this operation on the node's right subtree

3. Display the node

These three definitions provide distinct methods to characterize binary trees as an ordered list of nodes. Considering the example above, the pre-order traversal is F,B,A,D,C,E,G,I,H. The in-order traversal is A,B,C,D,E,F,G,H,I. The post-order traversal is A,C,E,D,B,H,I,G.

An important limitation of these traversals is that none of them uniquely define a binary tree.

Theorem 2.5. *The pre-, in-, and post-order traversal methods each do not uniquely define binary trees.*

Proof. What this claim means is that two binary trees could be represented by the same pre-, in-, or post-order traversal. This claim only asserts that one of the three traversals could be the same. To prove this, we need only find examples of two distinct binary trees with the same of one of these traversals.

Pre-Order Traversal: a chain from A to B to C is equivalent to A with a left leaf B and a right leaf C.

In-Order Traversal: a chain made by C with a left leaf B with a left leaf A is equivalent to B with a left leaf A and a right leaf C.

Post-Order Traversal: a chain made by A with a right leaf B with a right leaf C is equivalent to A with a left leaf C and a right leaf B. ∎

Next we will prove a stronger result before proving the result that will establish context for further study.

Theorem 2.6. *The combination of pre-order and post-order traversals does not uniquely define binary trees.*

Proof. Consider the two trees as follows: A with left leaf B with left leaf C, and A with right leaf B with right leaf C. The pre-order traversal of each of these is A,B,C. The post-order traversal of each of these is C,B,A. ∎

Theorem 2.7. *The combination of in-order traversal with either pre-order traversal or post-order traversal uniquely defines binary trees.*

Proof. Without loss of generality, we may consider the case of in-order traversal and pre-order traversal. Post-order traversal follows the same train of logic as pre-order traversal.

Consider two binary trees with the same pre-order traversal and the same in-order traversal. We seek to prove that the two binary trees are identical. First note that the trees must contain the same nodes. ∎