Tree Data Structure

Vivek K. S., Deepak G.

Information Systems Decision Sciences (ISDS)
MUMA College of Business
University of South Florida
Tampa, Florida

2017

Introduction to Trees

- Trees are a different type of a data structure unlike the Linear data structures like Stacks and Queues.
- Trees are used in many areas of computer science, including operating systems, graphics, database systems, and computer networking.
- Conceptually a tree is made up of a root, branches, and leaves.
- A document object model is an example for a tree data structure.

An HTML Object Tree

Properties of a Tree

The tree has the following properties.

- Each level of the tree represents a unique logical path in which the nodes are arranged in relation to one another.
- For example, all the information that pertains to the body of the html page comes under the html Node and not under the "Head" node.
- Second, the children of one node are different from another node's children even if they have the same names and may contain the same data.
- For example, in the html page, there could be muliplte ul ("unordered lists") elements each of them containing one or many li (List items) elements which are independent from one another.
- Third, each node is unique.
- For example, in the file system of your machine, each file and folder has a unique folder path that can be used to access that file or folder.

Taxonomy of a Tree

The following are some of the terms associated with a Tree.

- Node - The Node is the fundamental building block of a tree. It can have a name that we call the "key" and and an optional payload. The payload is the data contained by the node.
- Edge = The edge connects two nodes to show the relationships between the nodes. Except the root node, every other node in a tree has exactly one incoming edge.
- Each node may have several outgoing edges. In a Binary tree, each node only has two outgoing nodes.
- Root - It is the root of the tree and has no incoming edges.
- Path - The ordered list of nodes that are connected by nodes.

Continued..

- Children - The nodes that have incoming edges from the same node.
- Parent - The node that connects to all child nodes with edges.
- Sibling - All the nodes that share the same Parent node.
- Subtree - A subset of the tree comprised of a parent and all the descendants of that parent.
- Leaf Node - A node with no children of its own.
- Level - The level of a node 'n' is the number of edges on the path from the root node to 'n'.
- Height - The height of a tree is equal to the maximum level of any node in the tree.

Essential functions in a Tree

The Tree has the following essential functions. We are specifically going to discuss a binary tree.

- Ability to create new instances of a tree.
- Ability to get the subtree corresponding to the child node of the current node.
- Ability to create a new binary tree and install it as the child of the current node.
- Ability to set and get the value of the root of the tree.

The key decision in implementing a tree is choosing an internal storage technique.

Implementing a Tree in Python

In Python there are two ways to build a Tree,

- List of lists representation.
  - Each subtree is represented as a list of lists.
  - The list contains the value of the root as the first element and the left and right subtrees as lists themselves.
  - The nodes can be used by List indexing.
- Nodes and references technique.
  - We use an object-oriented paradigm with class representations for the root value, as well as the left and right subtrees.

Implementation in Python

We will explore the second method "Nodes and References" more in detail.
We can create a simple definition of the node as follows.

```
class BinaryTree:
        def __init__(self, root):
                self.key = root
                self.left_child = None
                self.right_child = None
```

In this definition, the constructor expects an object to be stored in the root value. We can store any type of data here. Lets store the character 'A' and the following alphabets in the child nodes so that it helps to improve our understanding of Trees.

Inserting a Left Child

To add a left child, we could create an instance of the Binary Tree and set the left child attribute of the root to point to this new object. Thus, it becomes a complete Binary Tree (subtree) of its own.

- When the left child does not exist, we create a new Binary Tree object and assign the value to it.
- When the left child already exists, we simply insert a node and push the existing child down one level in the tree.

```
def insert_left(self,new_node):
        if self.left_child == None:
            self.left_child = BinaryTree(new_node)
        else:
            temp = BinaryTree(new_node)
            temp.left_child = self.left_child
            self.left_child = temp
```

Inserting a Right Child

To add a right child, the method is the same.

```
def insert_right(self,new_node):
        if self.right_child == None:
            self.right_child = BinaryTree(new_node)
        else:
            temp = BinaryTree(new_node)
            temp.right_child = self.right_child
            self.right_child = temp
```

Accessor Methods to get the Child Nodes

```
def get_right_child(self):
        return self.right_child
def get_left_child(self):
        return self.left_child
def set_root_val(self,obj):
        self.key = obj
def get_root_val(self):
        return self.key
```

Summary

- We learned how trees differ from linear data structures and why they are important.
- We learned about nodes and their properties.
- We learned the taxonomy of a tree structure.
- We learned common tree operations and their implementations in Python.