POSTORDER TREE TRAVERSAL ALGORITHM IN PYTHON

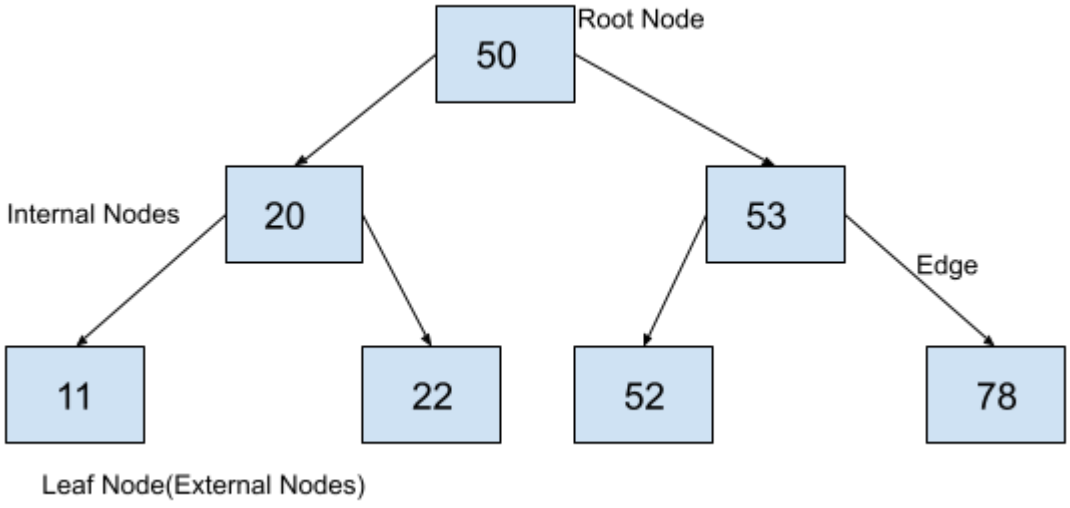
Binary trees are very useful in representing hierarchical data. In this article, we will discuss how to print all the elements in a binary tree using postorder tree traversal. We will also implement the postorder tree traversal algorithm in python.

What is the postorder tree traversal algorithm?

Postorder traversal algorithm is a depth first traversal algorithm. Here, we start from a root node and traverse a branch of the tree until we reach the end of the branch. After that, we move to the next branch. This process continues until all the nodes in the tree are printed.

The postorder tree traversal algorithm gets its name from the order in which the nodes of a tree are printed. In this algorithm, we first print the left sub-tree of the node, then we print the right sub-tree of the current node. At last, we print the current node. This process is recursive in nature. Here, the node is only printed when all the nodes in the left sub-tree and the right sub-tree of the current node have already been printed.

Let us understand the process using the binary tree given in the following image.



Binary tree

Let us print all of the nodes in the above binary tree using the postorder traversal algorithm.

We will start from the node 50. Before printing 50, we have to print its left sub-tree and right sub-tree. So, we will move to 20.

Before printing 20, we have to print its left sub-tree and right sub-tree. So, we will move to 11.

As 11 has no children, we will print 11. After that we will move to the previous node i.e. 20.

As the left child of 20 has already been printed, we will move to the right sub-tree of 20 i.e. 22.

As 22 has no children, we will print 22. After that we will move to the previous node i.e. 20.

As both the left sub-tree and the right sub-tree of 20 have already been printed. We will print 20 and will move to its parent node i.e. 50.

At this point, Left sub-tree of 50 has already been printed.So, will print its right sub-tree. We will move to 53.

Before printing 53, we have to print its left sub-tree and right sub-tree. So, we will move to 52.

As 52 has no children, we will print 52. After that we will move to the previous node i.e. 53.

As the left child of 53 has already been printed, we will move to the right sub-tree of 53 i.e. 78.

As 78 has no children, we will print 78. After that we will move to the previous node i.e. 53.

As both the left sub-tree and the right sub-tree of 53 have already been printed, We will print 53 and will move to its parent node i.e. 50.

At this point, both the left sub-tree and the right sub-tree of 50 have already been printed, So, we will print 50.

As all the nodes in the tree have already been printed, we will terminate this algorithm.

You can observe that we have printed the values in the order 11, 22, 20,52, 78, 53, 50. Let us now formulate an algorithm for the postorder tree traversal algorithm.

Algorithm for postorder tree traversal

As you have an overview of the entire process, we can formulate the algorithm for postorder tree traversal as follows.

1. Start from the root node.
2. If the root is empty, return.
3. Traverse the left sub-tree recursively.
4. Traverse the right sub-tree recursively.
5. Print the root node.
6. Stop.

Implementation of postorder tree traversal in Python

As we have understood the algorithm for postorder tree traversal and its working, Let us implement the algorithm and execute it for the binary tree given in the above image.

```
class BinaryTreeNode:
    def __init__(self, data):
        self.data = data
        self.leftChild = None
        self.rightChild = None
def postorder(root):
    # if root is None,return
    if root is None:
        return
    # traverse left subtree
    postorder(root.leftChild)
    # traverse right subtree
    postorder(root.rightChild)
    # print the current node
    print(root.data, end=" ,")
def insert(root, newValue):
    # if binary search tree is empty, create a new node and declare it as root
    if root is None:
        root = BinaryTreeNode(newValue)
        return root
    # if newValue is less than value of data in root, add it to left subtree and proceed recursively
    if newValue < root.data:
        root.leftChild = insert(root.leftChild, newValue)
    else:
        # if newValue is greater than value of data in root, add it to right subtree and proceed recursively
        root.rightChild = insert(root.rightChild, newValue)
    return root
```

```
root = insert(None, 50)
insert(root, 20)
insert(root, 53)
insert(root, 11)
insert(root, 22)
insert(root, 52)
insert(root, 78)
print("Postorder traversal of the binary tree is:")
postorder(root)
```

Output:

```
Postorder traversal of the binary tree is:
11 ,22 ,20 ,52 ,78 ,53 ,50 ,
```