IN-ORDER TREE TRAVERSAL IN PYTHON

You might have studied algorithms to traverse a python dictionary, a list, or a tuple. In this article, we will study the in-order traversal algorithm to traverse a binary tree. We will also discuss the implementation of the algorithm.

What is an In-order tree traversal algorithm?

In-order tree traversal algorithm is a depth first traversal algorithm. It means that we traverse the children of a node before traversing its siblings. In this way, we traverse to the maximum depth, print the elements till the last node and then come back to the top to print other elements.

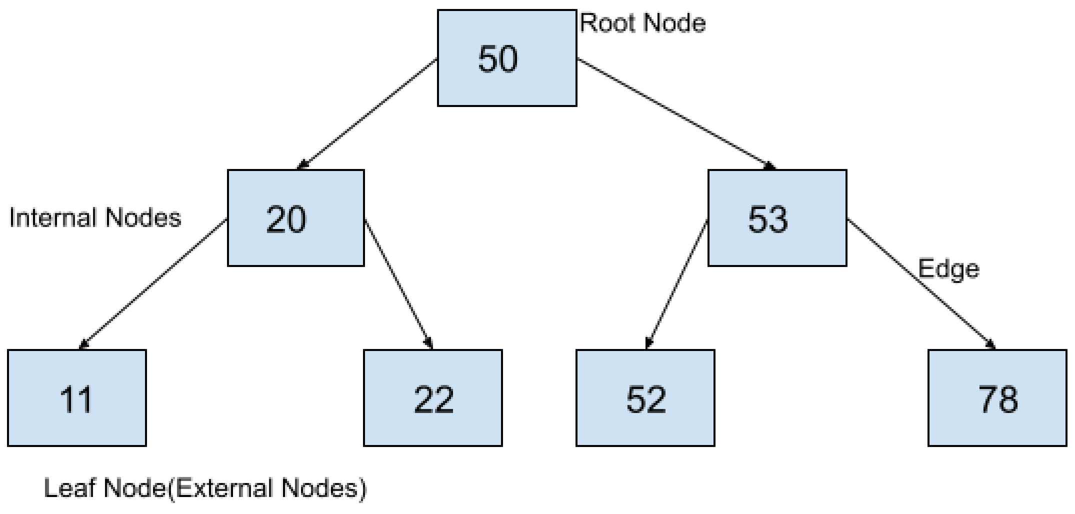
The algorithm is termed as in-order traversal algorithm because, when we traverse a binary search tree using this algorithm, The elements of the tree are printed in increasing order of their value.

Algorithm for In-order Traversal

We know that, In a binary search tree, The left child of any node contains an element less than the current node and the right child of a node contains an element greater than the current node. So, to print the elements in order, we will have to print the left child first, then the current node and the right child at last. This same rule will be used to print each node of the tree.

We will start from the root node, will traverse the left child or left sub-tree of the current node and then we will traverse the current node. At last we will traverse the right child or right sub-tree of the current node. We will perform this operation recursively till all the nodes are traversed.

To understand the concept better, let us use an example binary search tree and traverse it using the In-order binary tree traversal algorithm.



In-order Tree Traversal in Python

In-order Tree traversal

Here, we will start from root node 50. Before printing 50, we have to traverse the left subtree of 50. So, we will go to 20. Before printing 20, we have to traverse the left subtree of 20. So, we will go to 11. As 11 has no children, we will print 11 and move upwards to 20. After printing 20, we will traverse right subtree of 20. As 22 has no children, we will print 22 and move upwards to 50. At 50, its left subtree has been traversed, so we will print 50. After that we will traverse the right subtree of 50 using the same process. The in-order traversal of whole tree is : 11, 20, 22, 50, 52, 53, 78.

Implementation of In-order tree traversal in Python

The algorithm for in-order tree traversal can be formulated as follows.

1. Recursively Traverse the left sub-tree.
2. Print the root node.
3. Recursively Traverse the right sub-tree.

Here, we print the element in the node only if it is a leaf node or all the elements in its left subtree have already been printed.

We will now implement the above algorithm in python and will execute it for the binary search tree given in the above example.

```
class BinaryTreeNode:
  def __init__(self, data):
    self.data = data
    self.leftChild = None
    self.rightChild = None
def inorder(root):
  # if root is None,return
  if root == None:
    return
  # traverse left subtree
  inorder(root.leftChild)
  # print the current node
  print(root.data, end=" ,")
  # traverse right subtree
  inorder(root.rightChild)
```

```
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("Inorder traversal of the binary tree is:")
inorder(root)
```

Output:

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