Post-Order Traversal

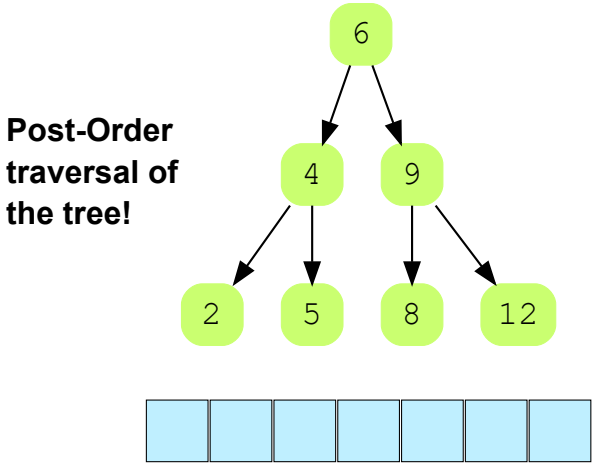
In this lesson, we will cover Post-Order Traversal in a Binary Search Tree and implement it in Python

We'll cover the following

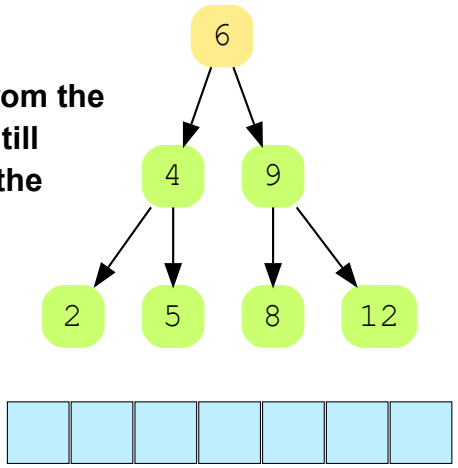
Introduction

In post-order traversal, the elements are traversed in “left-right-root” order. We first visit the left child, then the right child, and then finally the root/parent node. Here is a high-level description of the post-order traversal algorithm,

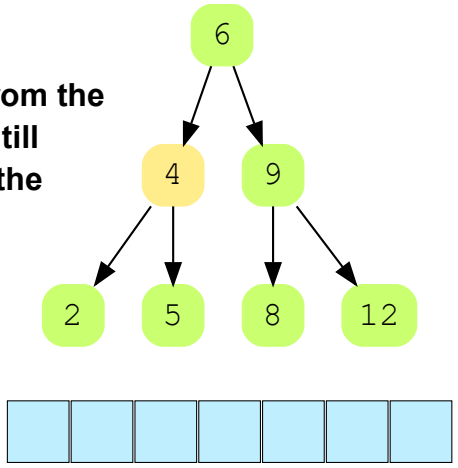
1. Traverse the left sub-tree of the ‘currentNode’ recursively by calling the `postOrderPrint()` function on it.
2. Traverse the right sub-tree of the ‘currentNode’ recursively by calling the `postOrderPrint()` function on it.
3. Visit current node and print its value



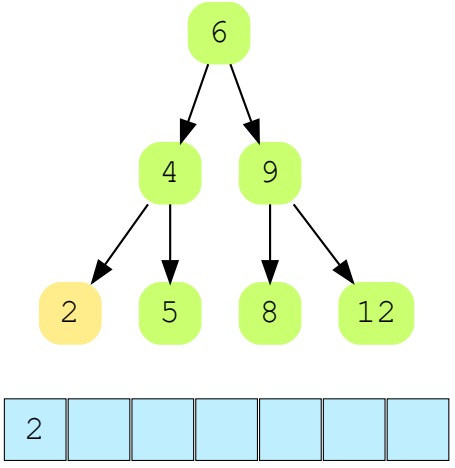
Traverse from the root to left till you reach the left leaf!



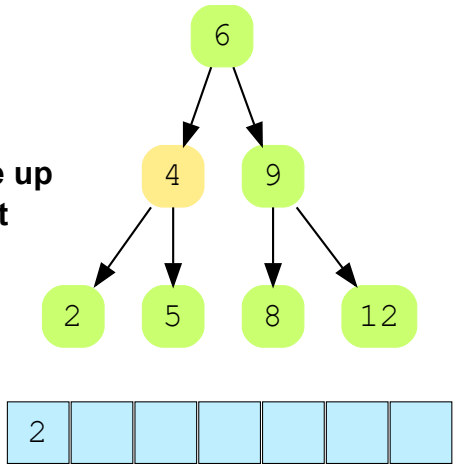
Traverse from the root to left till you reach the left leaf!



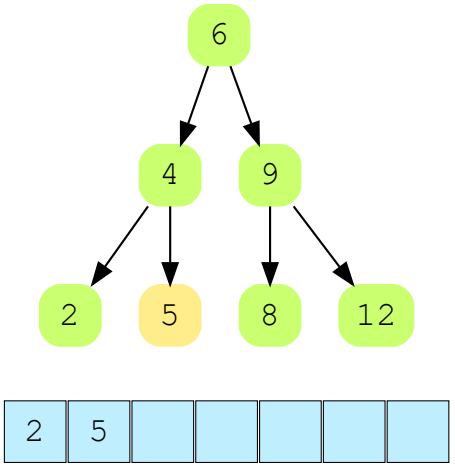
Now visit the left most leaf!



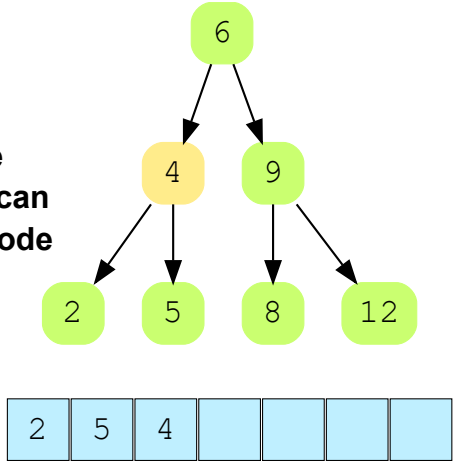
Traverse up to parent node!



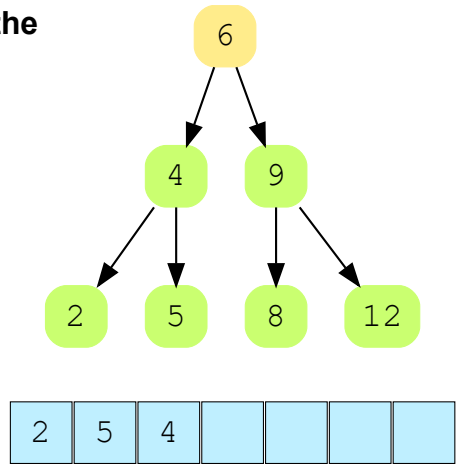
Now visit the right child!

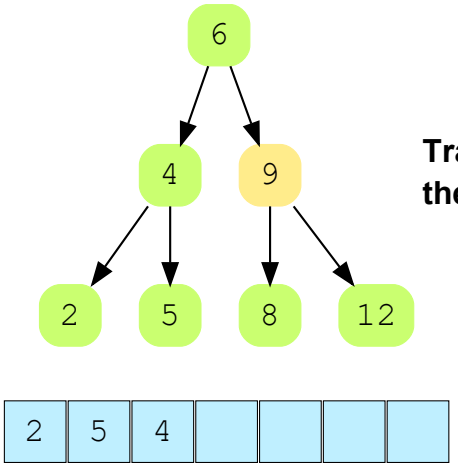


Now that both the children have been visited, we can visit the parent node

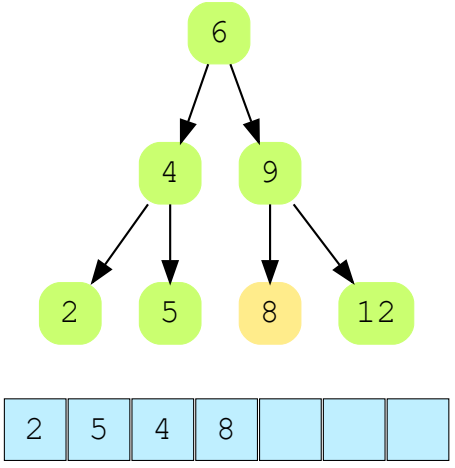


Traverse to the root again!

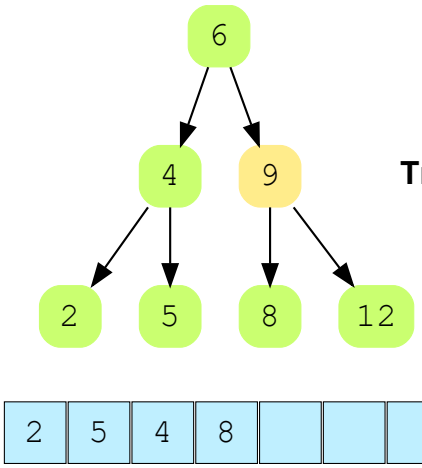




Traverse to the right node!

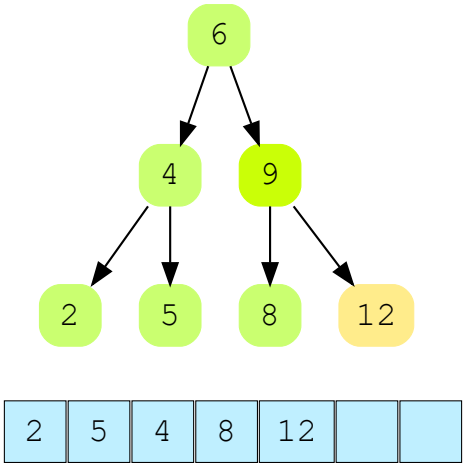


Now visit its left child first!



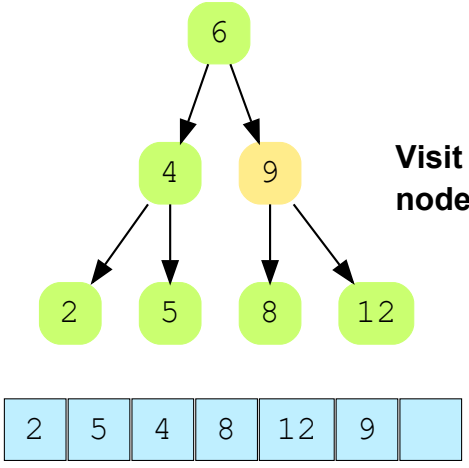
Traverse up!

11 of 15



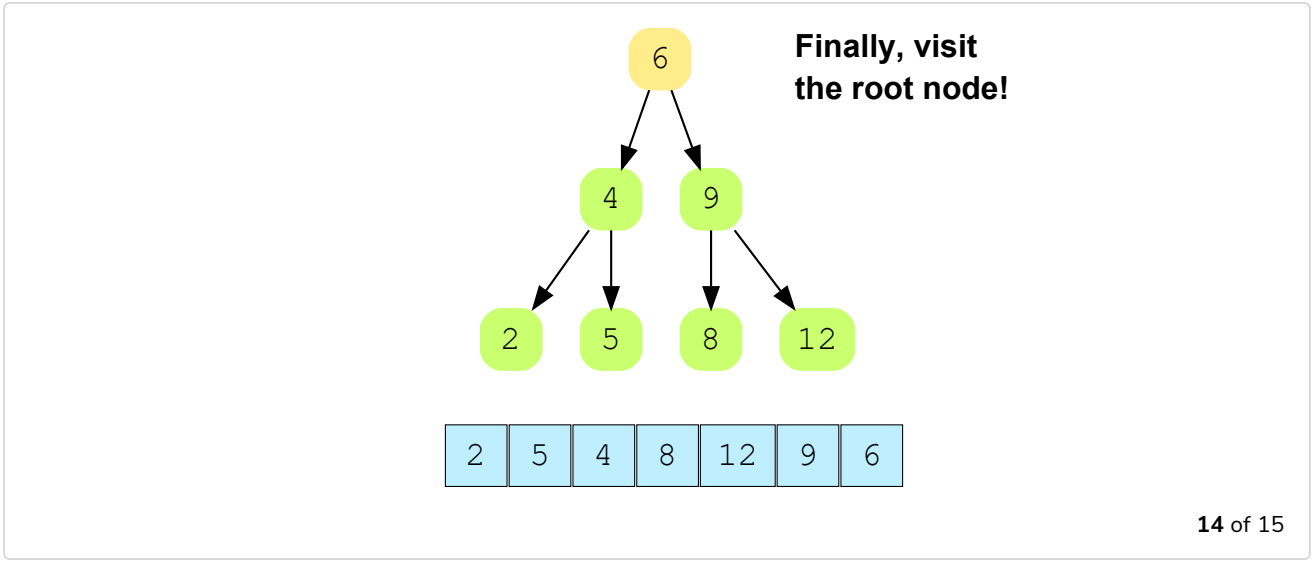
Visit its right child!

12 of 15

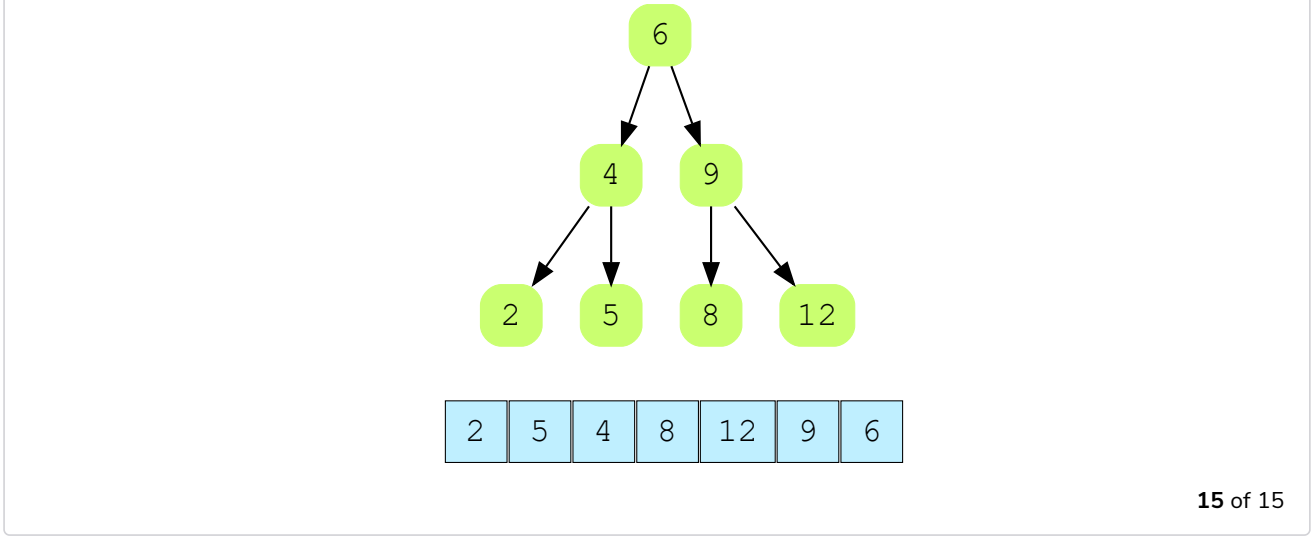


Visit the parent node now!

13 of 15



14 of 15



15 of 15

Implementation in Python

main.py

Node.py

BinarySearchTree.py

```
from Node import Node
from BinarySearchTree import BinarySearchTree


def postOrderPrint(node):
    if node is not None:
        postOrderPrint(node.leftChild)
        postOrderPrint(node.rightChild)
        print(node.val)


BST = BinarySearchTree(6)
BST.insert(4)
```

```
14  BST.insert(9)
15  BST.insert(5)
16  BST.insert(2)
17  BST.insert(8)
18  BST.insert(12)
19
20  postOrderPrint(BST.root)
21
```

```
Output                                                    0.194s

2
5
4
8
12
9
6
```

Explanation

First, we create an object of the `BinarySearchTree` class and insert some values into it. We then pass the tree's root to the `postOrderPrint()` function. If the node given is not `None`, this function calls `postOrderPrint()` on the left child first, then on the right child, and then finally prints the value at the node.

If you run the code for the BST above, it will print out the following

$$[2, 5, 4, 8, 12, 9, 6]$$

Looking at the code output, you can see that Post-Order traversal prints the children nodes first instead of the parent node.

In the next lesson, we'll study the final traversal strategy 'in-order traversal'.

Back

Pre-Order Traversal

Next

In-Order Traversal

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/post-order-traversal__introduction-to-trees__data-structures-for-coding-interviews-in-python)