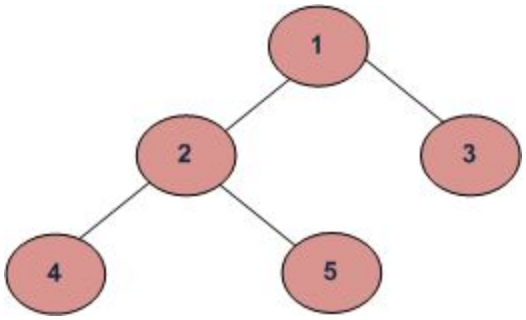
Date : 12th - Oct- 2020
Morning Session : 9am – 11.00 PM
By ~ Rohan Kumar

Topics: Binary Tree implementation

Tree Traversals (Inorder, Preorder and Postorder)

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.



Depth First Traversals:

(a) Inorder (Left, Root, Right) : 4 2 5 1 3

(b) Preorder (Root, Left, Right) : 1 2 4 5 3

(c) Postorder (Left, Right, Root) : 4 5 2 3 1

Breadth First or Level Order Traversal : 1 2 3 4 5

Inorder Traversal :

```
Algorithm Inorder(tree)
   1. Traverse the left subtree, i.e., call Inorder(left-subtree)
   2. Visit the root.
   3. Traverse the right subtree, i.e., call Inorder(right-subtree)
```

Uses of Inorder:

In case of binary search trees (BST), Inorder traversal gives nodes in non-decreasing order. To get nodes of BST in non-increasing order, a variation of Inorder traversal where Inorder traversal is reversed can be used.

Example: Inorder traversal for the above-given figure is 4 2 5 1 3.

Preorder Traversal

```
Algorithm Preorder(tree)
   1. Visit the root.
   2. Traverse the left subtree, i.e., call Preorder(left-subtree)
   3. Traverse the right subtree, i.e., call Preorder(right-subtree)
```

Uses of Preorder:

Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get a prefix expression on an expression tree.

Example: Preorder traversal for the above given figure is 1 2 4 5 3.

Postorder Traversal:

```
Algorithm Postorder(tree)
   1. Traverse the left subtree, i.e., call Postorder(left-subtree)
   2. Traverse the right subtree, i.e., call Postorder(right-subtree)
   3. Visit the root.
```

Uses of Postorder:

Postorder traversal is used to delete the tree.
Postorder traversal is also useful to get the postfix expression of an expression tree.

Example: Postorder traversal for the above given figure is 4 5 2 3 1.

Code Implementation:

```
class Node:
    def __init__(self,data):
        self.data = data
        self.left = None
        self.right = None

def inorder(root):
    if root:
        inorder(root.left)
        print(root.data,end = " ")
        inorder(root.right)

def preorder(root):
    if root:
        print(root.data,end = " ")
        preorder(root.left)
        preorder(root.right)

def postorder(root):
    if root:
        postorder(root.left)
        postorder(root.right)
        print(root.data,end = " ")
```

```

def search(root):
    if root:
        if root.data == key:
            print("found")
            return
        search(root.left)
        search(root.right)
```

```
32
33  def height(root):
34      if root is None:
35          return 0
36      leftHeight = height(root.left)
37      rightHeight = height(root.right)
38      if leftHeight > rightHeight:
39          return leftHeight + 1
40      else:
41          return rightHeight + 1
42
43  def checkSum(node):
44      left_data = 0
45      right_data = 0
46      if node == None or (node.left == None and node.right == None):
47          return 1
48      else:
49          if node.left!= None:
50              left_data = node.left.data
51          if node.right!= None:
52              right_data = node.right.data
53          if (node.data == left_data + right_data) and checkSum(node.left) and checkSum(node.right):
54              return 1
55          else:
56              return 0
```

```
58      root = Node(1)
59      root.left = Node(2)
60      root.right = Node(3)
61      root.left.left = Node(4)
62      root.left.right = Node(5)
63      root.right.left = Node(6)
64      root.right.right = Node(7)
65      inorder(root)
66      print()
67      preorder(root)
68      print()
69      postorder(root)
70      key = 4
71      print()
72      search(root)
73      print()
74      print(height(root))
```

Output:

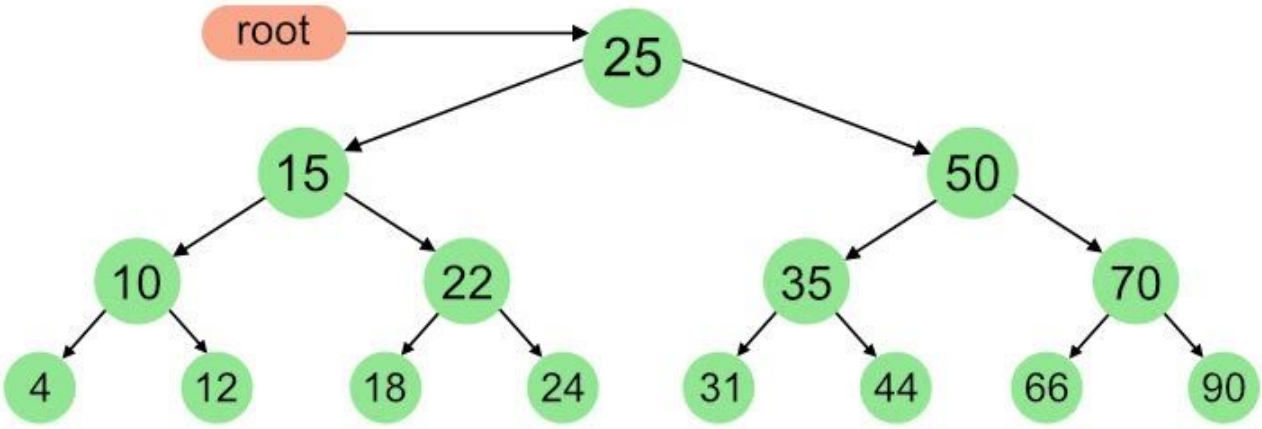
```
4 2 5 1 6 3 7
1 2 4 5 3 6 7
4 5 2 6 7 3 1
found

3
```

One More Example:

InOrder(root) visits nodes in the following order:
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Pre-order traversal visits nodes in the following order:
25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

A Post-order traversal visits nodes in the following order:
4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25



1. Order in inorder traversals
    a. Root->left child->right child
    b. Left child->right->child->root
    c. Left child->root->right child

Answer: **C**

2. Order in preorder traversals
    a. Root->left child->right child
    b. Left child->right->child->root
    c. Left child->root->right child

Answer: **A**

3. Order in postorder traversals
    a. Root->left child->right child
    b. Left child->right->child->root
    c. Left child->root->right child

Answer: **B**

4. Time complexity for finding height of binary tree
    a. O(log n)
    b. O(n)
    c. O(n*n)
    d. O(2^n)

Answer: **B**

To See Binary Tree Implementation please Go Through Recorded Lecture.

Recorded Lecture

Instructor Provided Resource Link:

https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/

Other Resource:

https://www.programiz.com/dsa/trees