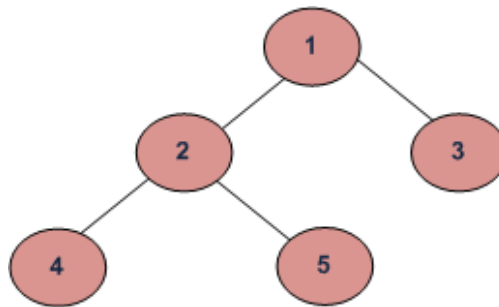


[Courses](#)[Login](#)[Suggest an Article](#)

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.



Example Tree

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

Please see [this](#) post for Breadth First Traversal.

Inorder Traversal (Practice):

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 s

reversed can be used.

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

فیلم الاكشن والاثارة
والتشويق

Preorder Traversal (Practice):

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 prefix expression on of an expression tree. Please see http://en.wikipedia.org/wiki/Polish_notation to know why prefix expressions are useful.

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

Postorder Traversal (Practice):

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. Please see [the question for deletion of tree](#) for details. Postorder traversal is also useful to get the postfix expression of an expression tree. Please see http://en.wikipedia.org/wiki/Reverse_Polish_notation to for the usage of postfix expression.

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

C++

```
// C program for different tree traversals
#include <iostream>
using namespace std;

/* A binary tree node has data, pointer to left child
and a pointer to right child */
```

```
struct Node
{
    int data;
    struct Node* left, *right;
    Node(int data)
    {
        this->data = data;
        left = right = NULL;
    }
};

/* Given a binary tree, print its nodes according to the
"bottom-up" postorder traversal. */
void printPostorder(struct Node* node)
{
    if (node == NULL)
        return;

    // first recur on left subtree
    printPostorder(node->left);

    // then recur on right subtree
    printPostorder(node->right);

    // now deal with the node
    cout << node->data << " ";
}

/* Given a binary tree, print its nodes in inorder*/
void printInorder(struct Node* node)
{
    if (node == NULL)
        return;

    /* first recur on left child */
    printInorder(node->left);

    /* then print the data of node */
    cout << node->data << " ";

    /* now recur on right child */
    printInorder(node->right);
}

/* Given a binary tree, print its nodes in preorder*/
void printPreorder(struct Node* node)
{
    if (node == NULL)
        return;

    /* first print data of node */
    cout << node->data << " ";

    /* then recur on left subtree */
    printPreorder(node->left);
```

```

    /* now recur on right subtree */
    printPreorder(node->right);
}

/* Driver program to test above functions*/
int main()
{
    struct Node *root = new Node(1);
    root->left          = new Node(2);
    root->right          = new Node(3);
    root->left->left      = new Node(4);
    root->left->right     = new Node(5);

    cout << "\nPreorder traversal of binary tree is \n";
    printPreorder(root);

    cout << "\nInorder traversal of binary tree is \n";
    printInorder(root);

    cout << "\nPostorder traversal of binary tree is \n";
    printPostorder(root);

    return 0;
}

```

C

```

// C program for different tree traversals
#include <stdio.h>
#include <stdlib.h>

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct node
{
    int data;
    struct node* left;
    struct node* right;
};

/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct node* newNode(int data)
{
    struct node* node = (struct node*)
        malloc(sizeof(struct node));

    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return(node);
}

/* Given a binary tree, print its nodes according to the

```

```
"bottom-up" postorder traversal. */
void printPostorder(struct node* node)
{
    if (node == NULL)
        return;

    // first recur on left subtree
    printPostorder(node->left);

    // then recur on right subtree
    printPostorder(node->right);

    // now deal with the node
    printf("%d ", node->data);
}

/* Given a binary tree, print its nodes in inorder*/
void printInorder(struct node* node)
{
    if (node == NULL)
        return;

    /* first recur on left child */
    printInorder(node->left);

    /* then print the data of node */
    printf("%d ", node->data);

    /* now recur on right child */
    printInorder(node->right);
}

/* Given a binary tree, print its nodes in preorder*/
void printPreorder(struct node* node)
{
    if (node == NULL)
        return;

    /* first print data of node */
    printf("%d ", node->data);

    /* then recur on left subtree */
    printPreorder(node->left);

    /* now recur on right subtree */
    printPreorder(node->right);
}

/* Driver program to test above functions*/
int main()
{
    struct node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
```

```
printf("\nPreorder traversal of binary tree is \n");
printPreorder(root);

printf("\nInorder traversal of binary tree is \n");
printInorder(root);

printf("\nPostorder traversal of binary tree is \n");
printPostorder(root);

getchar();
return 0;
}
```

Python

```
# Python program to for tree traversals

# A class that represents an individual node in a
# Binary Tree
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key

# A function to do inorder tree traversal
def printInorder(root):

    if root:

        # First recur on left child
        printInorder(root.left)

        # then print the data of node
        print(root.val),

        # now recur on right child
        printInorder(root.right)

# A function to do postorder tree traversal
def printPostorder(root):

    if root:

        # First recur on left child
        printPostorder(root.left)

        # the recur on right child
        printPostorder(root.right)
```

```
# now print the data of node
print(root.val),

# A function to do preorder tree traversal
def printPreorder(root):

    if root:

        # First print the data of node
        print(root.val),

        # Then recur on left child
        printPreorder(root.left)

        # Finally recur on right child
        printPreorder(root.right)

# Driver code
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
print "Preorder traversal of binary tree is"
printPreorder(root)

print "\nInorder traversal of binary tree is"
printInorder(root)

print "\nPostorder traversal of binary tree is"
printPostorder(root)
```

Java

```
// Java program for different tree traversals

/* Class containing left and right child of current
   node and key value*/
class Node
{
    int key;
    Node left, right;

    public Node(int item)
    {
        key = item;
        left = right = null;
    }
}

class BinaryTree
{
```

```
// Root of Binary Tree
Node root;

BinaryTree()
{
    root = null;
}

/* Given a binary tree, print its nodes according to the
"bottom-up" postorder traversal. */
void printPostorder(Node node)
{
    if (node == null)
        return;

    // first recur on left subtree
    printPostorder(node.left);

    // then recur on right subtree
    printPostorder(node.right);

    // now deal with the node
    System.out.print(node.key + " ");
}

/* Given a binary tree, print its nodes in inorder*/
void printInorder(Node node)
{
    if (node == null)
        return;

    /* first recur on left child */
    printInorder(node.left);

    /* then print the data of node */
    System.out.print(node.key + " ");

    /* now recur on right child */
    printInorder(node.right);
}

/* Given a binary tree, print its nodes in preorder*/
void printPreorder(Node node)
{
    if (node == null)
        return;

    /* first print data of node */
    System.out.print(node.key + " ");

    /* then recur on left subtree */
    printPreorder(node.left);

    /* now recur on right subtree */
    printPreorder(node.right);
}
```



```
// Wrappers over above recursive functions
void printPostorder() { printPostorder(root); }
void printInorder() { printInorder(root); }
void printPreorder() { printPreorder(root); }

// Driver method
public static void main(String[] args)
{
    BinaryTree tree = new BinaryTree();
    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);
    tree.root.left.left = new Node(4);
    tree.root.left.right = new Node(5);

    System.out.println("Preorder traversal of binary tree is ");
    tree.printPreorder();

    System.out.println("\nInorder traversal of binary tree is ");
    tree.printInorder();

    System.out.println("\nPostorder traversal of binary tree is ");
    tree.printPostorder();
}
}
```

C#

```
// C# program for different
// tree traversals
using System;

/* Class containing left and
right child of current
node and key value*/
class Node
{
    public int key;
    public Node left, right;

    public Node(int item)
    {
        key = item;
        left = right = null;
    }
}

class BinaryTree
{
    // Root of Binary Tree
    Node root;

    BinaryTree()
    {
        root = null;
    }
}
```

```
{
    root = null;
}

/* Given a binary tree, print
   its nodes according to the
   "bottom-up" postorder traversal. */
void printPostorder(Node node)
{
    if (node == null)
        return;

    // first recur on left subtree
    printPostorder(node.left);

    // then recur on right subtree
    printPostorder(node.right);

    // now deal with the node
    Console.Write(node.key + " ");
}

/* Given a binary tree, print
   its nodes in inorder*/
void printInorder(Node node)
{
    if (node == null)
        return;

    /* first recur on left child */
    printInorder(node.left);

    /* then print the data of node */
    Console.Write(node.key + " ");

    /* now recur on right child */
    printInorder(node.right);
}

/* Given a binary tree, print
   its nodes in preorder*/
void printPreorder(Node node)
{
    if (node == null)
        return;

    /* first print data of node */
    Console.Write(node.key + " ");

    /* then recur on left subtree */
    printPreorder(node.left);

    /* now recur on right subtree */
    printPreorder(node.right);
}
```

```
// Wrappers over above recursive functions
void printPostorder() {printPostorder(root);}
void printInorder() {printInorder(root);}
void printPreorder() {printPreorder(root);}

// Driver Code
static public void Main(String []args)
{
    BinaryTree tree = new BinaryTree();
    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);
    tree.root.left.left = new Node(4);
    tree.root.left.right = new Node(5);

    Console.WriteLine("Preorder traversal " +
                      "of binary tree is ");
    tree.printPreorder();

    Console.WriteLine("\nInorder traversal " +
                      "of binary tree is ");
    tree.printInorder();

    Console.WriteLine("\nPostorder traversal " +
                      "of binary tree is ");
    tree.printPostorder();
}
}

// This code is contributed by Arnab Kundu
```

Output:

```
Preorder traversal of binary tree is
1 2 4 5 3
Inorder traversal of binary tree is
4 2 5 1 3
Postorder traversal of binary tree is
4 5 2 3 1
```

One more example:

InOrder(root) visits nodes in the following order:

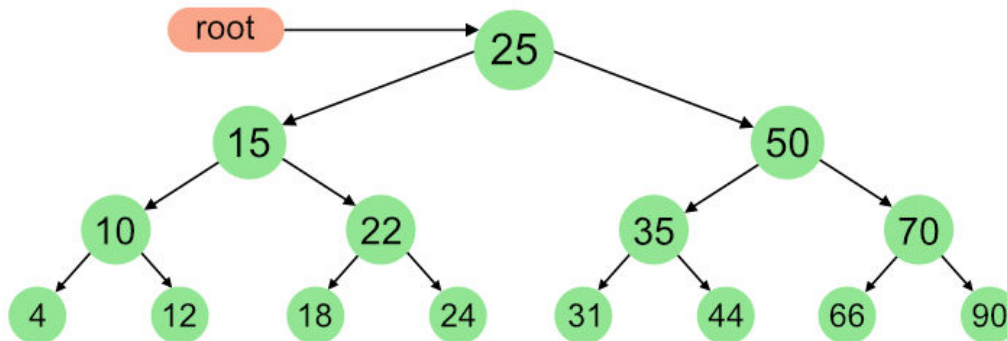
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

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



Time Complexity: $O(n)$

Let us see different corner cases.

Complexity function $T(n)$ – for all problem where tree traversal is involved – can be defined as:

$$T(n) = T(k) + T(n - k - 1) + c$$

Where k is the number of nodes on one side of root and $n-k-1$ on the other side.

Let's do an analysis of boundary conditions

Case 1: Skewed tree (One of the subtrees is empty and other subtree is non-empty)

k is 0 in this case.

$$T(n) = T(0) + T(n-1) + c$$

$$T(n) = 2T(0) + T(n-2) + 2c$$

$$T(n) = 3T(0) + T(n-3) + 3c$$

$$T(n) = 4T(0) + T(n-4) + 4c$$

.....

.....

$$T(n) = (n-1)T(0) + T(1) + (n-1)c$$

$$T(n) = nT(0) + (n)c$$

Value of $T(0)$ will be some constant say d . (traversing a empty tree will take some constants time)

$$T(n) = n(c+d)$$

$$T(n) = \Theta(n) \text{ (Theta of } n\text{)}$$

Case 2: Both left and right subtrees have equal number of nodes.

$$T(n) = 2T(\lfloor n/2 \rfloor) + c$$

This recursive function is in the standard form ($T(n) = aT(n/b) + (-)(n)$) for master method

http://en.wikipedia.org/wiki/Master_theorem. If we solve it by master method we get $\Theta(n)$

Auxiliary Space : If we don't consider size of stack for function calls then $O(1)$ otherwise $O(n)$.

Tree Traversals | GeeksforGeeks



Recommended Posts:

[Check if given Preorder, Inorder and Postorder traversals are of same tree](#)

[Preorder from Inorder and Postorder traversals](#)

[Print Postorder traversal from given Inorder and Preorder traversals](#)

Construct Tree from given Inorder and Preorder traversals

Construct Full Binary Tree from given preorder and postorder traversals

Construct a Binary Tree from Postorder and Inorder

Construct a tree from Inorder and Level order traversals | Set 2

Construct a tree from Inorder and Level order traversals | Set 1

Construct Full Binary Tree using its Preorder traversal and Preorder traversal of its mirror tree

Find postorder traversal of BST from preorder traversal

Construct a Binary Search Tree from given postorder

Postorder successor of a Node in Binary Tree

Find n-th node in Postorder traversal of a Binary Tree

Postorder traversal of Binary Tree without recursion and without stack

Postorder predecessor of a Node in Binary Search Tree

Improved By : danielbritten, andrew1234

Article Tags :

Articles

Tree

Inorder Traversal

PostOrder Traversal

Preorder Traversal

Snapdeal

tree-traversal

Trees

Practice Tags :

Snapdeal

Tree



35



To-do



Done

1.5

Based on **373** vote(s)

Feedback/ Suggest Improvement

Add Notes

Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!

A computer science portal for geeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org

COMPANY

About Us
Careers
Privacy Policy
Contact Us

PRACTICE

Company-wise
Topic-wise
Contests
Subjective Questions

LEARN

Algorithms
Data Structures
Languages
CS Subjects
Video Tutorials

CONTRIBUTE

Write an Article
Write Interview Experience
Internships
Videos

@geeksforgeeks, Some rights reserved