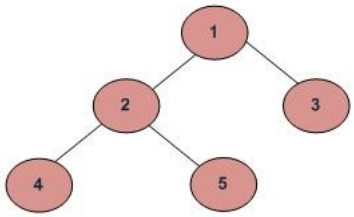
Binary Trees Traversals

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.



- Inorder (Left, Root, Right) : 4 2 5 1 3
- Preorder (Root, Left, Right) : 1 2 4 5 3.
- Postorder (Left, Right, Root) : 4 5 2 3 1

Lets look at each of these tree traversal algorithms in details:

- **Inorder Traversal :** In Inorder traversal a node is processed after processing of all nodes in its left subtree. The right subtree of the node is processed after processing the node itself.

```
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)
```

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

- **Preorder Traversal:** In preorder traversal a node is processed before processing any of the nodes in its subtree.

```
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)
```

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

- **Postorder Traversal:** In post order traversal, a node is processed after processing all of the nodes in its subtrees.

```
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.
```

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

C++ Java

```cpp
// 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;
    }
};
// Function to print the postorder traversal of a Binary Tree
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 << " ";
}
// Function to print the Inorder traversal of a Binary Tree
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);
}
// Function to print the PreOrder traversal of a Binary Tree
void printPreorder(struct Node* node)
{
    if (node == NULL)
        return;
    /* first print data of node */
    cout << node->data << " ";
    /* then recur on left sutree */
    printPreorder(node->left);
    /* now recur on right subtree */
    printPreorder(node->right);
}
// Driver Code
int main()
{
    // Contrust the Tree
    //      1
    //     /  \
```

```
57      //    2    3
58      //   / \
59      //  4   5
60
61      struct Node *root = new Node(1);
62      root->left    = new Node(2);
63      root->right    = new Node(3);
64      root->left->left = new Node(4);
65      root->left->right = new Node(5);
66
67      cout << "Preorder traversal of binary tree is \n";
68      printPreorder(root);
69
70      cout << "\nInorder traversal of binary tree is \n";
71      printInorder(root);
72
73      cout << "\nPostorder traversal of binary tree is \n";
74      printPostorder(root);
75
76      return 0;
77  }
```

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