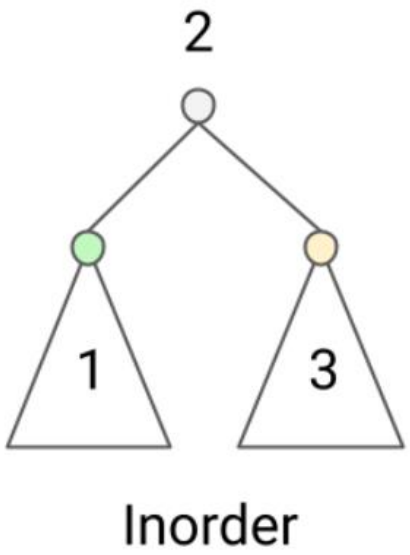
Data Structures and Algorithms

Lecture 26: Trees: In-order and Pre-order Traversal

In – Order Traversal

In-order traversal is defined as a type of tree traversal technique which follows the Left-Root-Right pattern, such that:

- The left subtree is traversed first
- Then the root node for that subtree is traversed
- Finally, the right subtre



Algorithm for In-order Traversal of Binary Tree

The algorithm for in-order traversal is shown as follows:

In-order(root):

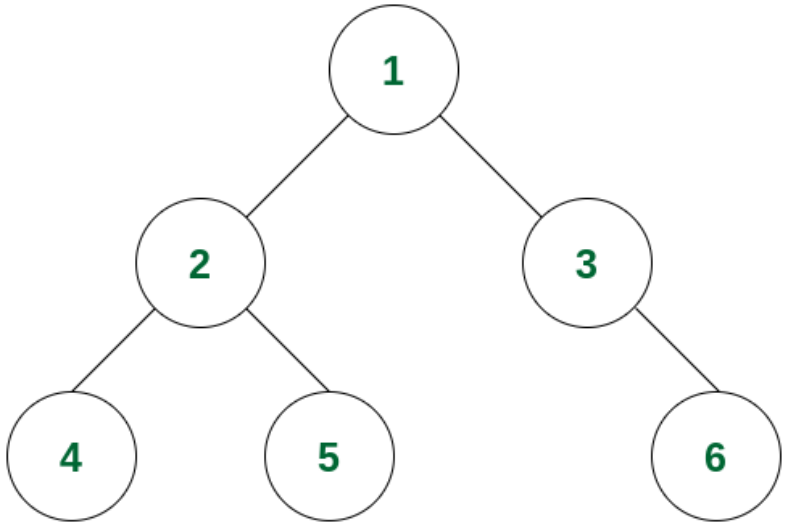
1.Follow step 2 to 4 until root != NULL

2.In-order (root -> left)

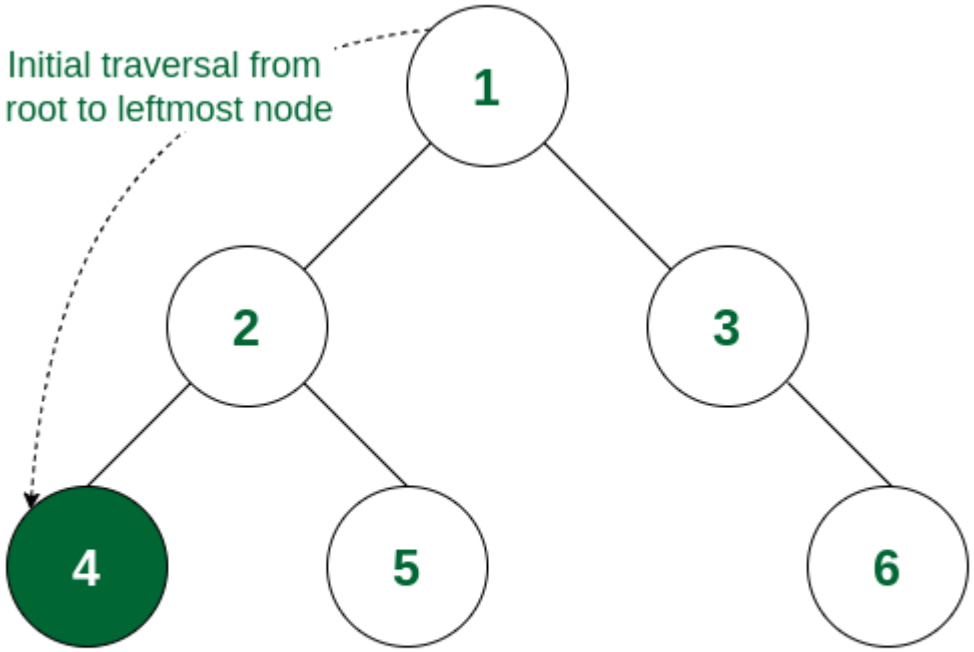
3.Write root -> dataAlgorithm for In-order Traversal of Binary Tree

4.The algorithm for in-order traversal is shown as follows:

5.Inorder(root):

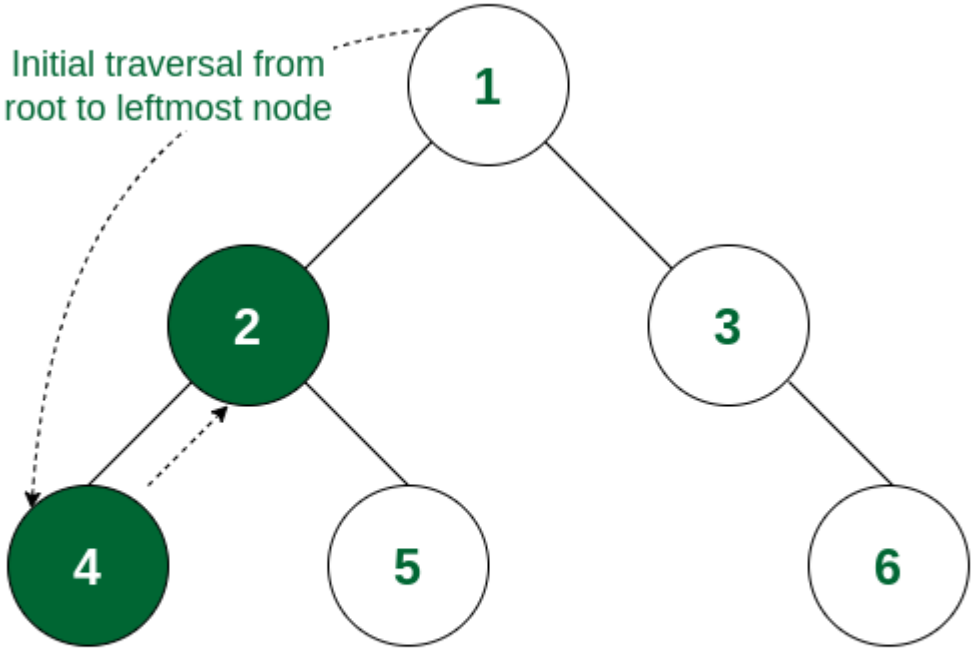


Step 1: The traversal will go from 1 to its left subtree i.e., 2, then from 2 to its left subtree root, i.e., 4. Now 4 has no left subtree, so it will be visited. It also does not have any right subtree. So no more traversal from 4



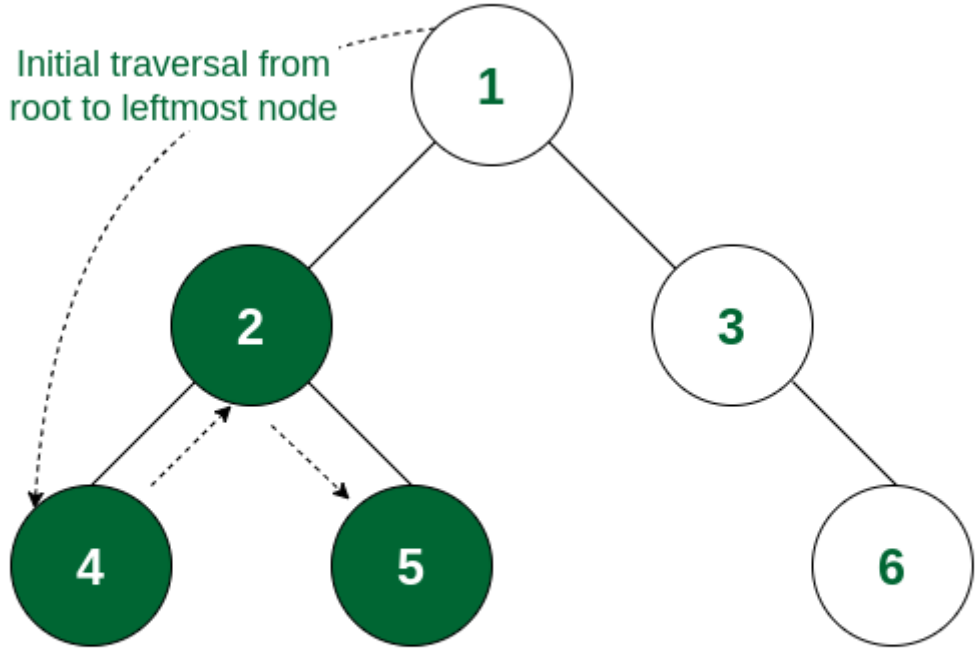
Leftmost node of the tree is visited

Step 2: As the left subtree of 2 is visited completely, now it read data of node 2 before moving to its right subtree.



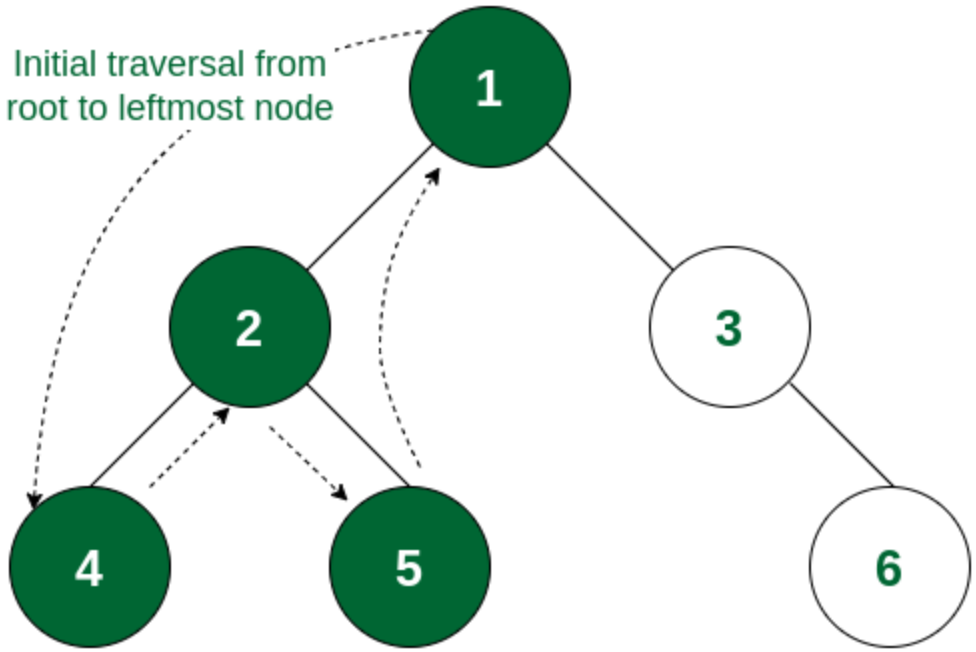
Left subtree of 2 is fully traversed. So 2 is visited next

Step 3: Now the right subtree of 2 will be traversed i.e., move to node 5. For node 5 there is no left subtree, so it gets visited and after that, the traversal comes back because there is no right subtree of node 5.



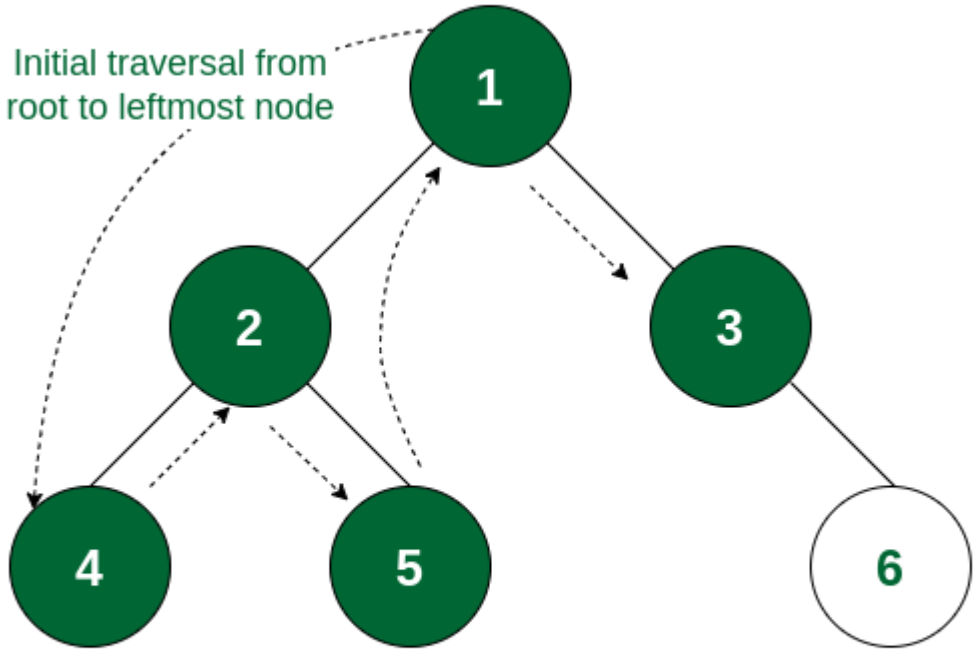
Right subtree of 2 (i.e., 5) is traversed

Step 4: As the left subtree of node 1 is, the root itself, i.e., node 1 will be visited.



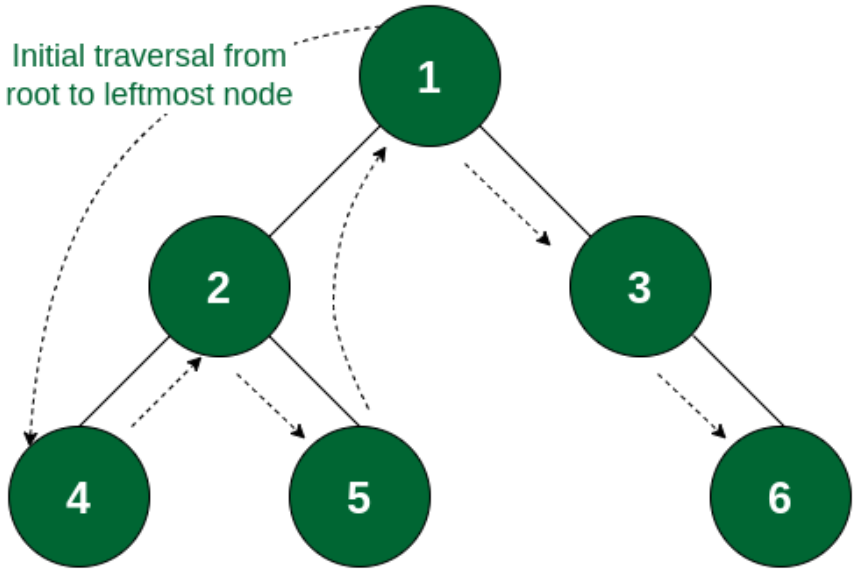
Left subtree of 1 is fully traversed. So 1 is visited next

Step 5: Left subtree of node 1 and the node itself is visited. So now the right subtree of 1 will be traversed i.e., move to node 3. As node 3 has no left subtree so it gets visited.



3 has no left subtree, so it is visited

Step 6: The left subtree of node 3 and the node itself is visited. So traverse to the right subtree and visit node 6. Now the traversal ends as all the nodes are traversed.



Right Child of 3 is visited

So the in-order of traversal of nodes is 4 -> 2 -> 5 -> 1 -> 3 -> 6.

```
function in_order(root, nodes) {
    if (root && root.left) {
        in_order(root.left, nodes);
    }
    nodes.push(root.data);
    if (root && root.right) {
        in_order(root.right, nodes);
    }
    return nodes;
}
```

```
// C++ program for inorder traversals

#include <bits/stdc++.h>
using namespace std;

// Structure of a Binary Tree Node
struct Node {
        int data;
        struct Node *left, *right;
        Node(int v)
        {
                data = v;
                left = right = NULL;
        }
};      // Function to print inorder traversal
        void printInorder(struct Node* node)
        {
                if (node == NULL)
                        return;

                // First recur on left subtree
                printInorder(node->left);

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

                // Then recur on right subtree
                printInorder(node->right);
        }

// Driver code
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);
        root->right->right = new Node(6);

        // Function call
        cout << "Inorder traversal of binary tree is: \n";
        printInorder(root);

        return 0;
}
```

Output

```
Inorder traversal of binary tree is:
4 2 5 1 3 6
```

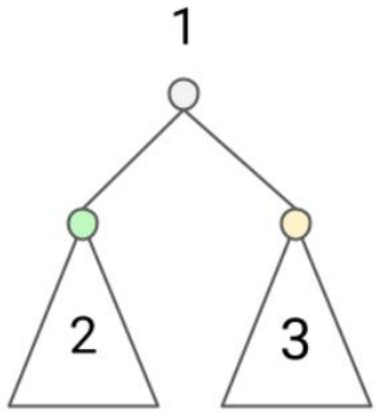
Use cases of In-order Traversal:

In the case of BST (Binary Search Tree), if any time there is a need to get the nodes in non-decreasing order, the best way is to implement an in-order traversal.

Pre – Order Traversal

Preorder traversal is defined as a type of tree traversal that follows the Root-Left-Right policy where:

- The root node of the subtree is visited first.
- Then the left subtree is traversed.
- At last, the right subtree is traversed.



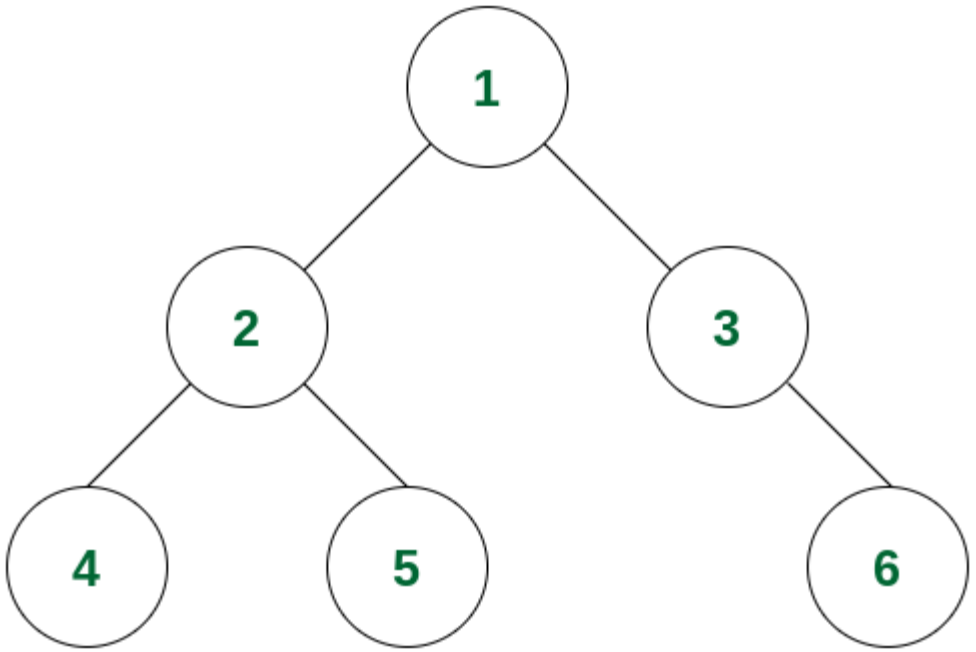
Preorder

Algorithm for In-order Traversal of Binary Tree

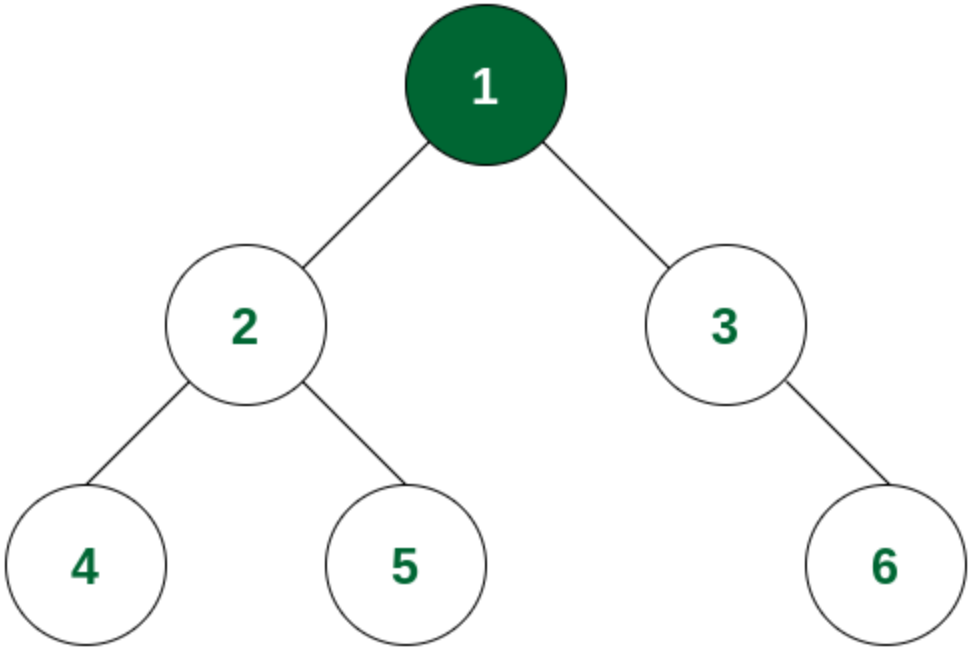
The algorithm for in-order traversal is shown as follows:

Preorder(root):

1. Follow step 2 to 4 until root != NULL
2. Write root -> data
3. Preorder (root -> left)
4. Preorder (root -> right)
5. End loop

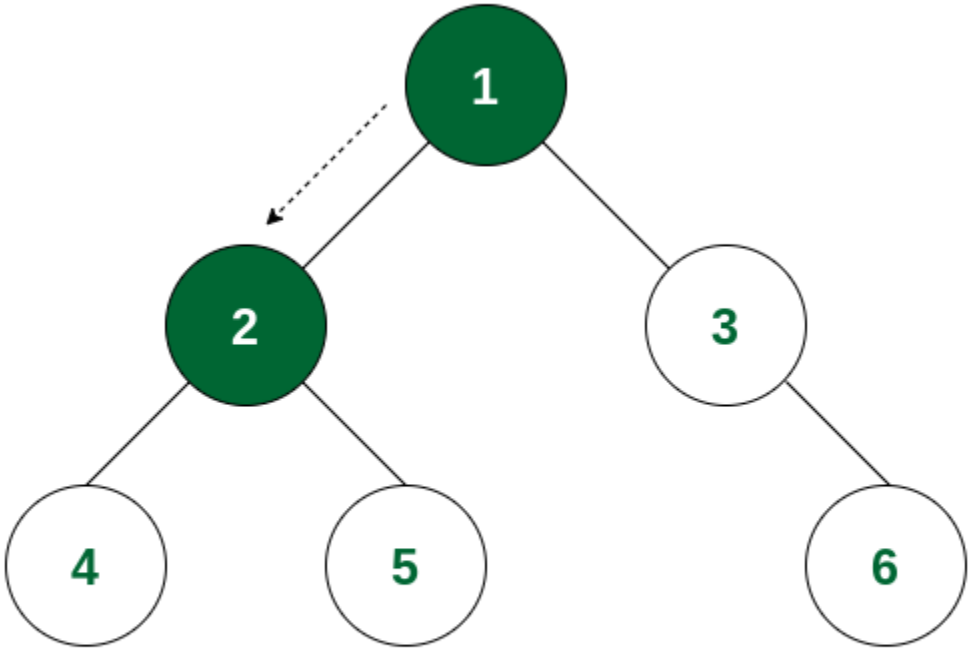


Step 1: *At first the root will be visited, i.e. node 1.*



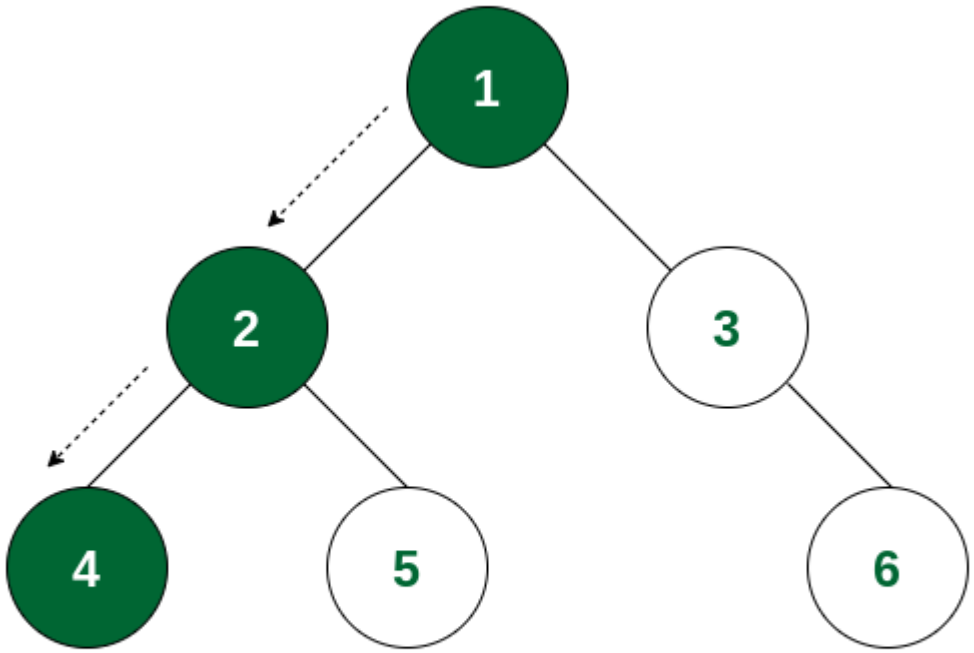
Root of the tree (i.e., 1) is visted

Step 2: *After this, traverse in the left subtree. Now the root of the left subtree is visited i.e., node 2 is visited.*



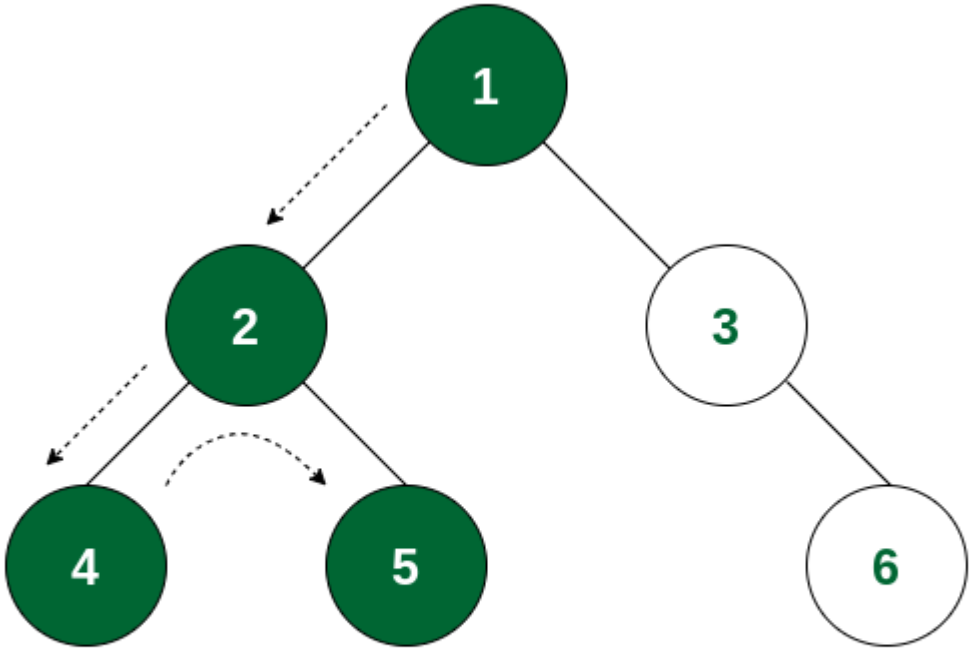
Root of left subtree of 1 (i.e., 2) is visited

Step 3: *Again the left subtree of node 2 is traversed and the root of that subtree i.e., node 4 is visited.*



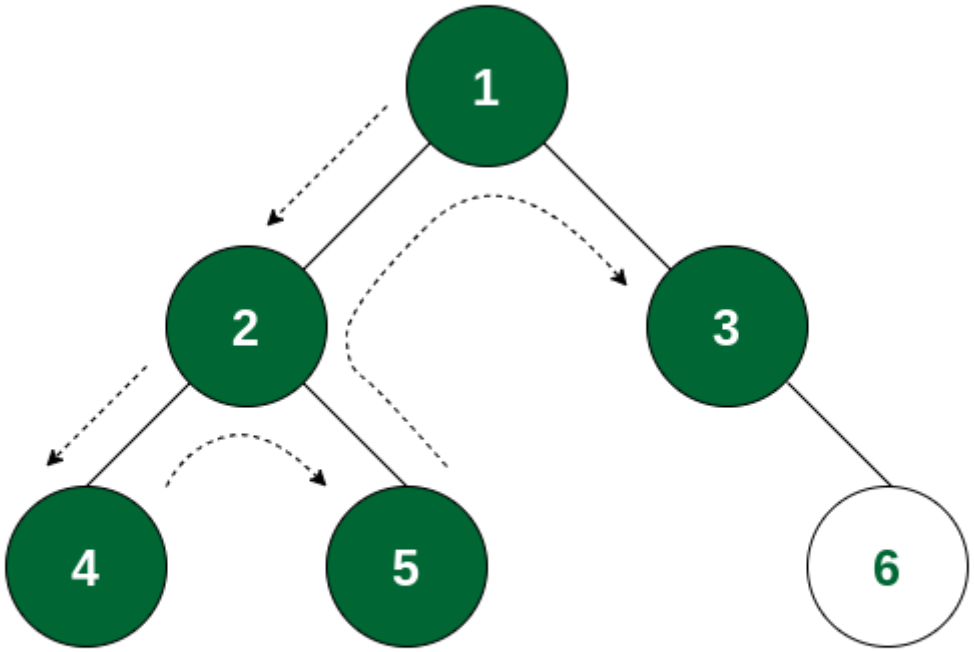
Left child of 2 (i.e., 4) is visited

Step 4: *There is no subtree of 4 and the left subtree of node 2 is visited. So now the right subtree of node 2 will be traversed and the root of that subtree i.e., node 5 will be visited.*



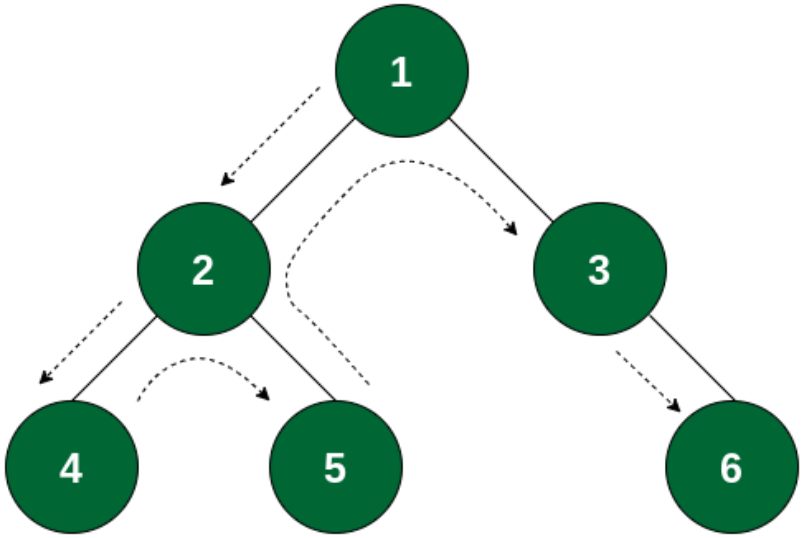
Right child of 2 (i.e., 5) is visited

Step 5: *The left subtree of node 1 is visited. So now the right subtree of node 1 will be traversed and the root node i.e., node 3 is visited.*



Root of right subtree of 1 (i.e., 3) is visited

Step 6: *Node 3 has no left subtree. So the right subtree will be traversed and the root of the subtree i.e., node 6 will be visited. After that there is no node that is not yet traversed. So the traversal ends.*



3 has no left subtree. So right subtree is visited

So the order of traversal of nodes is ***1 -> 2 -> 4 -> 5 -> 3 -> 6****.*

```
function pre_order(root, nodes) {
    nodes.push(root.data);
    if (root && root.left) {
        pre_order(root.left, nodes);
    }
    if (root && root.right) {
        pre_order(root.right, nodes);
    }
    return nodes;
}
```

```cpp
// C++ program for preorder traversals

#include <bits/stdc++.h>
using namespace std;

// Structure of a Binary Tree Node
struct Node {
    int data;
    struct Node *left, *right;
    Node(int v)
    {
        data = v;
        left = right = NULL;
    }
};
```

```cpp
// Function to print preorder traversal
void printPreorder(struct Node* node)
{
    if (node == NULL)
        return;

    // Deal with the node
    cout << node->data << " ";

    // Recur on left subtree
    printPreorder(node->left);

    // Recur on right subtree
    printPreorder(node->right);
}
```

```
// Driver code
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);
    root->right->right = new Node(6);

    // Function call
    cout << "Preorder traversal of binary tree is: \n";
    printPreorder(root);

    return 0;
}
```

Output

```
Preorder traversal of binary tree is:
1 2 4 5 3 6
```

Use cases of Preorder Traversal:

Some use cases of preorder traversal are:

- This is often used for creating a copy of a tree.
- It is also useful to get the prefix expression from an expression tree.