GeeksforGeeks
A computer science portal for geeks

Custom Search

COURSES

Login

HIRE WITH US

Inorder Tree Traversal without recursion and without stack!

Using Morris Traversal, we can traverse the tree without using stack and recursion. The idea of Morris Traversal is based on Threaded Binary Tree. In this traversal, we first create links to Inorder successor and print the data using these links, and finally revert the changes to restore original tree.

```
1. Initialize current as root
2. While current is not NULL
   If the current does not have left child
      a) Print current’s data
      b) Go to the right, i.e., current = current->right
   Else
      a) Make current as the right child of the rightmost
         node in current's left subtree
      b) Go to this left child, i.e., current = current->left
```

Although the tree is modified through the traversal, it is reverted back to its original shape after the completion. Unlike Stack based traversal, no extra space is required for this traversal.

C++

```
#include <stdio.h>
#include <stdlib.h>
```

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

/* Function to traverse the binary tree without recursion and
   without stack */
void MorrisTraversal(struct tNode* root)
{
    struct tNode *current, *pre;

    if (root == NULL)
        return;

    current = root;
    while (current != NULL) {

        if (current->left == NULL) {
            printf("%d ", current->data);
            current = current->right;
        }
        else {

            /* Find the inorder predecessor of current */
            pre = current->left;
            while (pre->right != NULL && pre->right != current)
                pre = pre->right;

            /* Make current as the right child of its inorder
               predecessor */
            if (pre->right == NULL) {
                pre->right = current;
                current = current->left;
            }

            /* Revert the changes made in the 'if' part to restore
               the original tree i.e., fix the right child
               of predecessor */
            else {
                pre->right = NULL;
                printf("%d ", current->data);
                current = current->right;
            } /* End of if condition pre->right == NULL */
        } /* End of if condition current->left == NULL*/
    } /* End of while */
}

/* UTILITY FUNCTIONS */
/* Helper function that allocates a new tNode with the
   given data and NULL left and right pointers. */
```

```
struct tNode* newtNode(int data)
{
    struct tNode* node = new tNode;
    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return (node);
}

/* Driver program to test above functions*/
int main()
{

    /* Constructed binary tree is
            1
          /   \
        2      3
      /  \
    4     5
  */
    struct tNode* root = newtNode(1);
    root->left = newtNode(2);
    root->right = newtNode(3);
    root->left->left = newtNode(4);
    root->left->right = newtNode(5);

    MorrisTraversal(root);

    return 0;
}
```

Java

```
// Java program to print inorder traversal without recursion and stack

/* A binary tree tNode has data, a pointer to left child
   and a pointer to right child */
class tNode {
    int data;
    tNode left, right;

    tNode(int item)
    {
        data = item;
        left = right = null;
    }
}

class BinaryTree {
    tNode root;
```

```
/* Function to traverse a binary tree without recursion and
   without stack */
void MorrisTraversal(tNode root)
{
    tNode current, pre;

    if (root == null)
        return;

    current = root;
    while (current != null) {
        if (current.left == null) {
            System.out.print(current.data + " ");
            current = current.right;
        }
        else {
            /* Find the inorder predecessor of current */
            pre = current.left;
            while (pre.right != null && pre.right != current)
                pre = pre.right;

            /* Make current as right child of its inorder predecessor */
            if (pre.right == null) {
                pre.right = current;
                current = current.left;
            }

            /* Revert the changes made in the 'if' part to restore the
                original tree i.e., fix the right child of predecessor*/
            else {
                pre.right = null;
                System.out.print(current.data + " ");
                current = current.right;
            } /* End of if condition pre->right == NULL */

        } /* End of if condition current->left == NULL*/

    } /* End of while */
}

public static void main(String args[])
{
    /* Constructed binary tree is
           1
         /   \
        2      3
      /   \
    4     5
    */
    BinaryTree tree = new BinaryTree();
    tree.root = new tNode(1);
    tree.root.left = new tNode(2);
    tree.root.right = new tNode(3);
    tree.root.left.left = new tNode(4);
```

```
        tree.root.left.right = new tNode(5);

        tree.MorrisTraversal(tree.root);
    }
}

// This code has been contributed by Mayank Jaiswal(mayank_24)
```

Python

```
# Python program to do inorder traversal without recursion and
# without stack Morris inOrder Traversal

# A binary tree node
class Node:

    # Constructor to create a new node
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# Iterative function for inorder tree traversal
def MorrisTraversal(root):

    # Set current to root of binary tree
    current = root

    while(current is not None):

        if current.left is None:
            print current.data,
            current = current.right
        else:
            # Find the inorder predecessor of current
            pre = current.left
            while(pre.right is not None and pre.right != current):
                pre = pre.right

            # Make current as right child of its inorder predecessor
            if(pre.right is None):
                pre.right = current
                current = current.left

            # Revert the changes made in if part to restore the
            # original tree i.e., fix the right child of predecessor
            else:
                pre.right = None
                print current.data,
                current = current.right

# Driver program to test the above function
```

```
"""
Constructed binary tree is
            1
          /   \
        2      3
      /  \
    4     5
"""
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)

MorrisTraversal(root)

# This code is contributed by Naveen Aili
```

C#

```
// C# program to print inorder traversal
// without recursion and stack
using System;

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

class BinaryTree
{
    tNode root;

public class tNode
{
    public int data;
    public tNode left, right;

    public tNode(int item)
    {
        data = item;
        left = right = null;
    }
}
    /* Function to traverse binary tree without
     recursion and without stack */
    void MorrisTraversal(tNode root)
    {
        tNode current, pre;

        if (root == null)
            return;
        current = root;
        while (current != null)
        {
            if (current.left == null)
            {
                Console.Write(current.data + " ");
                current = current.right;
            }
            else
            {
                /* Find the inorder predecessor of current */
                pre = current.left;
                while (pre.right != null && pre.right != current)
                    pre = pre.right;

                /* Make current as right child
                of its inorder predecessor */
                if (pre.right == null)
                {
                    pre.right = current;
                    current = current.left;
                }

                /* Revert the changes made in
                if part to restore the original
                tree i.e., fix the right child
                of predecssor*/
                else
                {
                    pre.right = null;
                    Console.Write(current.data + " ");
                    current = current.right;
                } /* End of if condition pre->right == NULL */

            } /* End of if condition current->left == NULL*/

        } /* End of while */
    }

    // Driver code
    public static void Main(String []args)
    {
        /* Constructed binary tree is
            1
            / \
            2     3
        / \
        4     5
        */
        BinaryTree tree = new BinaryTree();
        tree.root = new tNode(1);
        tree.root.left = new tNode(2);
        tree.root.right = new tNode(3);
        tree.root.left.left = new tNode(4);
        tree.root.left.right = new tNode(5);

        tree.MorrisTraversal(tree.root);
    }
}

// This code has been contributed
// by Arnab Kundu
```

Output:

```
 4 2 5 1 3
```

Time Complexity : O(n) If we take a closer look, we can notice that every edge of the tree is traversed at most two times. And in the worst case, the same number of extra edges (as input tree) are created and removed.

References:
www.liacs.nl/~deutz/DS/september28.pdf
www.scss.tcd.ie/disciplines/software_systems/…/HughGibbonsSlides.pdf

Please write comments if you find any bug in above code/algorithm, or want to share more information about stack Morris Inorder Tree Traversal.

Recommended Posts:

Inorder Non-threaded Binary Tree Traversal without Recursion or Stack

Inorder Tree Traversal without Recursion

Postorder traversal of Binary Tree without recursion and without stack

Cartesian tree from inorder traversal | Segment Tree

Construct Special Binary Tree from given Inorder traversal

Calculate height of Binary Tree using Inorder and Level Order Traversal

DFS traversal of a tree using recursion

Preorder Traversal of N-ary Tree Without Recursion

Zig-Zag traversal of a Binary Tree using Recursion

Find maximum and minimum element in binary tree without using recursion or stack or queue

Find n-th node of inorder traversal

Find all possible binary trees with given Inorder Traversal

Print Postorder traversal from given Inorder and Preorder traversals

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

Iterative Postorder Traversal | Set 2 (Using One Stack)

Improved By : andrew1234, Akanksha_Rai

Article Tags : Tree threaded-binary-tree tree-traversal

Practice Tags : Tree

47

4.2

□ To-do □ Done

Based on **504** 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

▲

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

LEARN
Algorithms
Data Structures
Languages
CS Subjects
Video Tutorials

PRACTICE
Courses
Company-wise
Topic-wise
How to begin?

CONTRIBUTE
Write an Article
Write Interview Experience
Internships
Videos

▲