Monday, March 14, 2022 9:33 PM

Inorder Tree Traversal without recursion and without stack!

- Difficulty Level : Hard
- Last Updated : 26 Oct, 2021

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) Find rightmost node in current left subtree OR
            node whose right child == current.
         If we found right child == current
            a) Update the right child as NULL of that node whose
right child is current
            b) Print current's data
            c) Go to the right, i.e. current = current->right
         Else
            a) Make current as the right child of that rightmost
               node we found; and
            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 */
}

// 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 Mayank
// Jaiswal(mayank_24)
```

Python 3

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


class Node:
    """A binary tree node"""

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


def morris_traversal(root):
    """Generator function for
      iterative inorder tree traversal"""

    current = root

    while current is not None:

        if current.left is None:
            yield current.data
            current = current.right
        else:

            # Find the inorder
            # predecessor of current
            pre = current.left
            while pre.right is not None
                  and pre.right is not current:
                pre = pre.right

            if pre.right is None:

                # Make current as right
                # child of its inorder predecessor
                pre.right = current
                current = current.left

            else:
                # Revert the changes made
```

```
                # in the 'if' part to restore the
                # original tree. i.e., fix
                # the right child of predecessor
                pre.right = None
                yield current.data
                current = current.right


# Driver code
"""
Constructed binary tree is
            1
          /   \
         2     3
       /   \
      4     5
"""
root = Node(1,
            right=Node(3),
            left=Node(2,
                      left=Node(4),
                      right=Node(5)
                      )
            )

for v in morris_traversal(root):
    print(v, end=' ')

# This code is contributed by Naveen Aili
# updated by Elazar Gershuni
```

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 predecessor*/
            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
```

Javascript

```
<script>

// JavaScript 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
{
    constructor(item)
    {
        this.data = item;
        this.left = this.right = null;
    }
}

let root;

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

        if (root == null)
            return;

        current = root;
        while (current != null)
        {
            if (current.left == null)
            {
                document.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 the 'if' part
                   to restore the original
                   tree i.e., fix
                   the right child of predecessor*/
                else
                {
                    pre.right = null;
                    document.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
/* Constructed binary tree is
               1
             /   \
            2      3
          /   \
         4     5
        */

root = new tNode(1);
root.left = new tNode(2);
root.right = new tNode(3);
root.left.left = new tNode(4);
root.left.right = new tNode(5);

MorrisTraversal(root);

// This code is contributed by avanitrachhadiya2155
```

```
</script>
```

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 three times. And in the worst case, the same number of extra edges (as input tree) are created and removed.

From <https://www.geeksforgeeks.org/inorder-tree-traversal-without-recursion-and-without-stack/>