Morris traversal for Preorder

Using Morris Traversal, we can traverse the tree without using stack and recursion. The algorithm for Preorder is almost similar to Morris traversal for Inorder.

1...**If** left child is null, print the current node data. Move to right child.

….**Else**, Make the right child of the inorder predecessor point to the current node. Two cases arise:

……**a)** The right child of the inorder predecessor already points to the current node. Set right child to NULL. Move to right child of current node.

……**b)** The right child is NULL. Set it to current node. Print current node's data and move to left child of current node.

2...Iterate until current node is not NULL.

Following is the implementation of the above algorithm.

C

```
// C program for Morris Preorder traversal
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int data;
    struct node *left, *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* temp = (struct node*) malloc(sizeof(struct node));
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// Preorder traversal without recursion and without stack
void morrisTraversalPreorder(struct node* root)
{
    while (root)
```

```
    {
        // If left child is null, print the current node data. Move to
        // right child.
        if (root->left == NULL)
        {
            printf( "%d ", root->data );
            root = root->right;
        }
        else
        {
            // Find inorder predecessor
            struct node* current = root->left;
            while (current->right && current->right != root)
                current = current->right;

            // If the right child of inorder predecessor already points to
            // this node
            if (current->right == root)
            {
                current->right = NULL;
                root = root->right;
            }

            // If right child doesn't point to this node, then print this
            // node and make right child point to this node
            else
            {
                printf("%d ", root->data);
                current->right = root;
                root = root->left;
            }
        }
    }
}

// Function for sStandard preorder traversal
void preorder(struct node* root)
{
    if (root)
    {
        printf( "%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

/* Driver program to test above functions*/
int main()
{
    struct node* root = NULL;

    root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);

    root->left->left = newNode(4);
    root->left->right = newNode(5);

    root->right->left = newNode(6);
    root->right->right = newNode(7);

    root->left->left->left = newNode(8);
    root->left->left->right = newNode(9);

    root->left->right->left = newNode(10);
```

```
    root->left->right->right = newNode(11);

    morrisTraversalPreorder(root);

    printf("\n");
    preorder(root);

    return 0;
}
```

Run on IDE

Java

```
// Java program to implement Morris preorder traversal

// A binary tree node
class Node {

    int data;
    Node left, right;

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

class BinaryTree {

    Node root;

    void morrisTraversalPreorder()
    {
        morrisTraversalPreorder(root);
    }

    // Preorder traversal without recursion and without stack
    void morrisTraversalPreorder(Node node) {
        while (node != null) {

            // If left child is null, print the current node data. Move to
            // right child.
            if (node.left == null) {
                System.out.print(node.data + " ");
                node = node.right;
            } else {

                // Find inorder predecessor
                Node current = node.left;
                while (current.right != null && current.right != node) {
                    current = current.right;
                }

                // If the right child of inorder predecessor already points to
                // this node
                if (current.right == node) {
                    current.right = null;
                    node = node.right;
                }

                // If right child doesn't point to this node, then print this
```

```
                // node and make right child point to this node
                else {
                    System.out.print(node.data + " ");
                    current.right = node;
                    node = node.left;
                }
            }
        }
    }

    void preorder()
    {
        preorder(root);
    }

    // Function for Standard preorder traversal
    void preorder(Node node) {
        if (node != null) {
            System.out.print(node.data + " ");
            preorder(node.left);
            preorder(node.right);
        }
    }

    // Driver programs to test above functions
    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);
        tree.root.right.left = new Node(6);
        tree.root.right.right = new Node(7);
        tree.root.left.left.left = new Node(8);
        tree.root.left.left.right = new Node(9);
        tree.root.left.right.left = new Node(10);
        tree.root.left.right.right = new Node(11);
        tree.morrisTraversalPreorder();
        System.out.println("");
        tree.preorder();

    }
}

// this code has been contributed by Mayank Jaiswal
```

Run on IDE

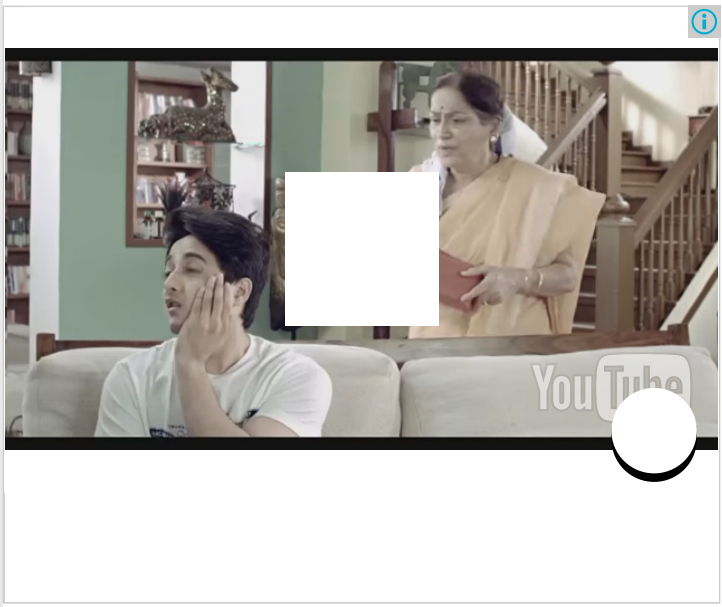
Output:

```
1 2 4 8 9 5 10 11 3 6 7
1 2 4 8 9 5 10 11 3 6 7
```

Limitations:

Morris traversal modifies the tree during the process. It establishes the right links while moving down the tree and resets the right links while moving up the tree. So the algorithm cannot be applied if write operations are not allowed.

This article is compiled by **Aashish Barnwal** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



23 Comments Category: Trees

Related Posts:

- Check if removing an edge can divide a Binary Tree in two halves
- Check sum of Covered and Uncovered nodes of Binary Tree
- Lowest Common Ancestor in a Binary Tree | Set 2 (Using Parent Pointer)
- Construct a Binary Search Tree from given postorder
- BFS vs DFS for Binary Tree
- Maximum difference between node and its ancestor in Binary Tree
- Inorder Non-threaded Binary Tree Traversal without Recursion or Stack
- Check if leaf traversal of two Binary Trees is same?

(Login to Rate and Mark)

3.6 Average Difficulty : **3.6/5.0**
Based on **5** vote(s)

Add to TODO List
Mark as DONE

Like Share 13 people like this.

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

23 Comments GeeksforGeeks 1 Login

Recommend 3 Share Sort by Newest

Join the discussion…

AS • 8 months ago
for inorder what we did was made the left subtree's rightmost node's right child point to current node. As that's where the current should have appeared in inorder traversal. Here won't printing current and then making left subtree's rightmost node's right child point to right subtree be enough?
• Reply • Share ›

Dman • 9 months ago
Easily implemented with little modifications in indorder morris traversal

https://ideone.com/Kxoqlf
1 • Reply • Share ›

Guest • a year ago
isn't this inorder successor actually? the code looks to be going to inorder successor. I mean its going to rightmost of left child.
• Reply • Share ›

DS+Algo ➚ Guest • 10 months ago
rightmost of left child of node is inorder predecessor,
inorder traversal print kara aur check kar le
• Reply • Share ›

Mr. Lazy • a year ago
Completely same as inorder, only differs in placement of print statement in else part.
4 • Reply • Share ›

Vinod • a year ago
Time Complexity is O(n). Each edge is traversed at most 3 times and there are n-1 edges in a tree, hence the O(3(n-1)) = O(n).
• Reply • Share ›

Vikram Ojha • a year ago
My code is bit faster as compared to given code: this runs for 14 time for same input while the given function is 22 times for same input. so we can say its bit optimized.

```
int preorderTraversalWithoutStack(struct Node *root,int &count){
/* Base Condition */
if(root==NULL)
return 0;
```

```
return 0;
struct Node *current,*pre;
current=root;
while(current!=NULL)
cout<<current->info<<" ";

/* if leftsubtree is NULL print current node and move right */
if(current->lc==NULL)
current=current->rc;

else{
pre=current->lc;
while(pre->rc!=NULL)
```

see more

• Reply • Share ›

green_ginger • 2 years ago

hey,

am new to Morris tree traversal and I tried to write pre-order in my own way..

I doubt my algorithm, as it is SLIGHTLY different from the above, whether it covered all possible cases,

I will be thankful if you guys evaluate the code below..

```
void morrisTraversalPreorder(struct node* root)
{
struct node* current = root;
while (current)
{
printf( "%d ", current->data );
if (current->left == NULL)
{

current = current->right;
```

see more

• Reply • Share ›

<HoldOnLife!#> • 2 years ago

Is morris traversal traversal better than using stack in the previous method?

• Reply • Share ›

Kim Jong-il <HoldOnLife!#> • 2 years ago

No. It takes more time than that

1 • Reply • Share ›

np • 2 years ago

By changing just one printf statement from preorder we can convert it to inorder

following is te link for inorder

http://ideone.com/G5Ar4J

happy coding

1 • Reply • Share ›

gastonbm • 2 years ago

Great! It's almost the same, it was hard to notice the difference :P

2 • Reply • Share ›

AlienOnEarth • 2 years ago

Excellent

• Reply • Share ›

X • 2 years ago

A correction in the if conditions for current.right != null:

In the traversal, if current.right == root, it means we have reached a node that's already processed(threaded) and this can be printed and the link reverted back to null. If current.right == null, it means we are reaching this node for this first time and this needs to linked to the inorder successor. So, I think the conditions should be swapped to the following. Please correct me if I 'am wrong:

```
if(current.right == null)
{
current.right = root;
root = root.left;
}
else
{
print(current.data);
current.right = null;
root = root.right;
}
```

• Reply • Share ›

sap • 3 years ago

what about morris traversal for postorder?

abhishek08aug • 3 years ago

Intelligent :D

• Reply • Share ›

Soumya Sengupta • 3 years ago

why is it printing the tree twice??

```
/* Paste your code here (You may delete these lines if not writing code) */
```

1 • Reply • Share ›

Debjit • 3 years ago

Viky,

Linear time complexity and constant space complexity

```
/* Paste your code here (You may delete these lines if not writing code) */
```

• Reply • Share ›

ubiquitous ➚ Debjit • 3 years ago

this is not linear time complexity. it's O(NLogN) and constant space. So sometimes the memory restriction is so much that this traversal helps in great time rathe then O(N) recursive preorder traversal. Hope it helps.

2 • Reply • Share ›

Madhav • 3 years ago

Great Solution.
Thanks.

1 • Reply • Share ›

Viky • 3 years ago

What is the complexity of this algorithm?

• Reply • Share ›

Saurabh Jain ➚ Viky • 2 years ago

Time Complexity -> n Log(n)

where n -> number of nodes.

As
you can see there are two while loops.First while loop is iterating
through each and every node and second while loop is helping to find the

inorder predecessor .

1. Due to outer loop -> O(n)

2. Due to inner loop -> O(Log(n))

• Reply • Share ›

Shyam Raj • 3 years ago

This is really an interesting algorithm :).

Had to debug a lot to figure out what&#039s really happening.

But loved it indeed :).

• Reply • Share ›