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.

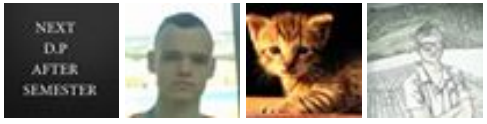
1...**If** left child is null, print the current node data. Move to right child.
....**Else**, Make the right child of the inorder predecessor point to the current node. Two cases arise:
.....**a)** The right child of the inorder predecessor already points to the current node. Set right child to NULL. 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 C implementation of the above algorithm.

```
// 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 point
            // this node
            if (current->right == root)
            {
                current->right = NULL;
                root = root->right;
            }

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

Geometric Algorithms

Popular Posts

```
    {
        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;
}
```

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.

Related Tpoics:

- Print a Binary Tree in Vertical Order | Set 2 (Hashmap based Method)
- Print Right View of a Binary Tree
- Red-Black Tree | Set 3 (Delete)
- Construct a tree from Inorder and Level order traversals
- Print all nodes at distance k from a given node
- Print a Binary Tree in Vertical Order | Set 1
- Interval Tree
- Check if a given Binary Tree is height balanced like a Red-Black Tree

Recent Comments

affiszerv Your example has two 4s on row 3, that's why it...

Backtracking | Set 7 (Sudoku) · 35 minutes ago

RVM Can someone please elaborate this Qs from above...

Flipkart Interview | Set 6 · 55 minutes ago

Vishal Gupta I talked about as an Interviewer in general,...

Software Engineering Lab, Samsung Interview | Set 2 · 55 minutes ago

@meya Working solution for question 2 of 4f2f round....

Amazon Interview | Set 53 (For SDE-1) · 1 hour ago

sandeep void rearrange(struct node *head) {...

Given a linked list, reverse alternate nodes and append at the end · 2 hours ago

Neha I think that is what it should return as, in...

Find depth of the deepest odd level leaf node · 3 hours ago

Writing code in comment? Please use **ideone.com** and share the link here.

11 Comments GeeksforGeeks

Sort by Newest

Join the discussion…

AlienOnEarth • 3 days ago
Excellent
• Reply • Share ›

X • 2 months 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' this can be printed and the link reverted back to null. If current.right == null, it m this first time and this needs to linked to the inorder successor. So, I think 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 • 11 months ago

what about morris traversal for postorder?

Reply • Share ›

abhishek08aug • 11 months ago

Intelligent :D

Reply • Share ›

Soumya Sengupta • a year ago

why is it printing the tree twice??

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

Reply • Share ›

Debjit • a year ago

Viky,

Linear time complexity and constant space complexity

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

Reply • Share ›

ubiquitous ➦ Debjit • 9 months ago

this is not linear time complexity. it's O(NLogN) and constant space. S
so much that this traversal helps in great time rathe then O(N) recursiv

1 ^ | v • Reply • Share ›

Madhav • a year ago

Great Solution.
Thanks.

1 ^ | v • Reply • Share ›

Viky • a year ago

What is the complexity of this algorithm?

^ | v • Reply • Share ›

Nitin Sharma ➦ Viky • 4 months 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 t inorder predecessor .

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

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

^ | v • Reply • Share ›

Shyam Raj • a year ago

This is really an interesting algorithm :).

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

But loved it indeed :).

^ | v • Reply • Share ›