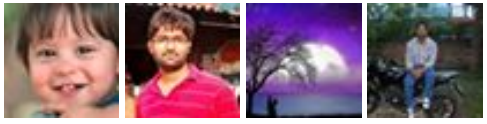
Iterative Postorder Traversal | Set 1 (Using Two Stacks)

We have discussed iterative inorder and iterative preorder traversals. In this post, iterative postorder traversal is discussed which is more complex than the other two traversals (due to its nature of non-tail recursion, there is an extra statement after the final recursive call to itself). The postorder traversal can easily be done using two stacks though. The idea is to push reverse postorder traversal to a stack. Once we have reverse postorder traversal in a stack, we can just pop all items one by one from the stack and print them, this order of printing will be in postorder because of LIFO property of stacks. Now the question is, how to get reverse post order elements in a stack – the other stack is used for this purpose. For example, in the following tree, we need to get 1, 3, 7, 6, 2, 5, 4 in a stack. If take a closer look at this sequence, we can observe that this sequence is very similar to preorder traversal. The only difference is right child is visited before left child and therefore sequence is “root right left” instead of “root left right”. So we can do something like iterative preorder traversal with following differences.
a) Instead of printing an item, we push it to a stack.
b) We push left subtree before right subtree.

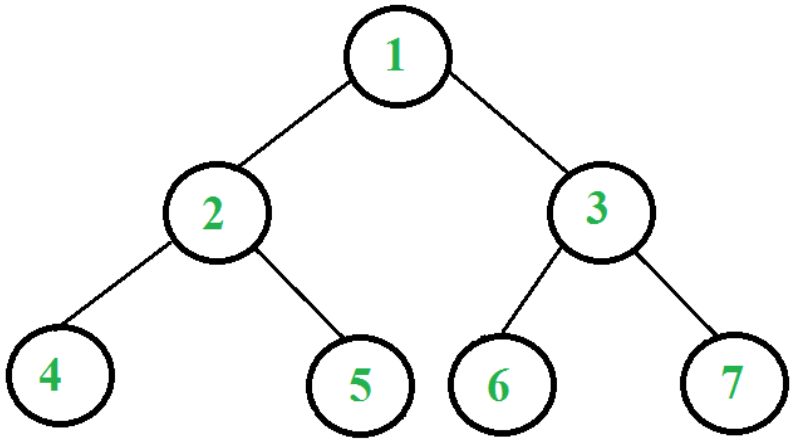
Following is the complete algorithm. After step 2, we get reverse postorder traversal in second stack. We use first stack to get this order.

```
1. Push root to first stack.
2. Loop while first stack is not empty
   2.1 Pop a node from first stack and push it to second stack
   2.2 Push left and right children of the popped node to first stack
3. Print contents of second stack
```

Let us consider the following tree



Interview Experiences
Advanced Data Structures
Dynamic Programming
Greedy Algorithms
Backtracking
Pattern Searching
Divide & Conquer
Mathematical Algorithms
Recursion



Following are the steps to print postorder traversal of the above tree using two stacks.

```
1. Push 1 to first stack.
      First stack: 1
      Second stack: Empty

2. Pop 1 from first stack and push it to second stack.
   Push left and right children of 1 to first stack
      First stack: 2, 3
      Second stack: 1

3. Pop 3 from first stack and push it to second stack.
   Push left and right children of 3 to first stack
      First stack: 2, 6, 7
      Second stack:1, 3

4. Pop 7 from first stack and push it to second stack.
      First stack: 2, 6
      Second stack:1, 3, 7

5. Pop 6 from first stack and push it to second stack.
      First stack: 2
      Second stack:1, 3, 7, 6
```

Geometric Algorithms

Popular Posts

All permutations of a given string
Memory Layout of C Programs
Understanding “extern” keyword in C
Median of two sorted arrays
Tree traversal without recursion and without stack!
Structure Member Alignment, Padding and Data Packing
Intersection point of two Linked Lists
Lowest Common Ancestor in a BST.
Check if a binary tree is BST or not
Sorted Linked List to Balanced BST

```
6. Pop 2 from first stack and push it to second stack.
   Push left and right children of 2 to first stack
      First stack: 4, 5
      Second stack:1, 3, 7, 6, 2

7. Pop 5 from first stack and push it to second stack.
      First stack: 4
      Second stack: 1, 3, 7, 6, 2, 5

8. Pop 4 from first stack and push it to second stack.
      First stack: Empty
      Second stack: 1, 3, 7, 6, 2, 5, 4

The algorithm stops since there is no more item in first stack.
Observe that content of second stack is in postorder fashion. Print them.
```

Following is C implementation of iterative postorder traversal using two stacks.

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

// Maximum stack size
#define MAX_SIZE 100

// A tree node
struct Node
{
    int data;
    struct Node *left, *right;
};

// Stack type
struct Stack
{
    int size;
    int top;
    struct Node* *array;
};

// A utility function to create a new tree node
struct Node* newNode(int data)
```

```
{
    struct Node* node = (struct Node*) malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

// A utility function to create a stack of given size
struct Stack* createStack(int size)
{
    struct Stack* stack =
             (struct Stack*) malloc(sizeof(struct Stack));
    stack->size = size;
    stack->top = -1;
    stack->array =
           (struct Node**) malloc(stack->size * sizeof(struct Node*));
    return stack;
}

// BASIC OPERATIONS OF STACK
int isFull(struct Stack* stack)
{  return stack->top - 1 == stack->size; }

int isEmpty(struct Stack* stack)
{  return stack->top == -1;  }

void push(struct Stack* stack, struct Node* node)
{
    if (isFull(stack))
        return;
    stack->array[++stack->top] = node;
}

struct Node* pop(struct Stack* stack)
{
    if (isEmpty(stack))
        return NULL;
    return stack->array[stack->top--];
}

// An iterative function to do post order traversal of a given binary
void postOrderIterative(struct Node* root)
{
    // Create two stacks
    struct Stack* s1 = createStack(MAX_SIZE);
    struct Stack* s2 = createStack(MAX_SIZE);
```


Recent Comments

```
    // push root to first stack
    push(s1, root);
    struct Node* node;

    // Run while first stack is not empty
    while (!isEmpty(s1))
    {
        // Pop an item from s1 and push it to s2
        node = pop(s1);
        push(s2, node);

        // Push left and right children of removed item to s1
        if (node->left)
            push(s1, node->left);
        if (node->right)
            push(s1, node->right);
    }

    // Print all elements of second stack
    while (!isEmpty(s2))
    {
        node = pop(s2);
        printf("%d ", node->data);
    }
}

// Driver program to test above functions
int main()
{
    // Let us construct the tree shown in above figure
    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);

    postOrderIterative(root);

    return 0;
}
```

Output:

```
4 5 2 6 7 3 1
```

Following is overview of the above post.
Iterative preorder traversal can be easily implemented using two stacks. The first stack is used to get the reverse postorder traversal in second stack. The steps to get reverse postorder are similar to iterative preorder.

You may also like to see a method which uses only one stack.

This article is compiled by Aashish Barnwal. 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

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

8 Comments GeeksforGeeks

Sort by Newest

Naruto • 3 months ago
incorrect Condition for stackfull
• Reply • Share ›

abhishek • 10 months ago
In isFull function instead of stack->top-1==stack->size, it should be stack->top
top is pointing to the index of array of hundred elements starting with zero and 100th element .
correct me if i am wrong
1 • Reply • Share ›

Bin abhishek • a month ago
exactly !
• Reply • Share ›

Anonymous • 11 months ago
Java code for Post order Traversal of Binary Tree

```
 public void IpostOrder(Node rt)
{
  Stack s1=new Stack();
```

```
 Stack s2=new Stack();

Node temp;

 s1.push(rt);

 while(!s1.isEmpty())
 {
     temp=(Node) s1.pop();

     s2.push(temp);
     if(temp.left!=null)
         s1.push(temp.left);
```

see more

• Reply • Share ›

abhishek08aug • 11 months ago

Intelligent :D

• Reply • Share ›

Nagaraju • a year ago

Another approach, simulated how recursion is doing, Please correct me if i an

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

public static void iterativePostOrder(Node root)
   {
       Stack<Node> st1 = new Stack<Node>();
       Stack<Node> st2 = new Stack<Node>();
       st1.push(root);
       Node current = root;
```

```
boolean visitedLeaf = false;
while(!st1.isEmpty())
{
    Node t1 = st1.peek();
    Node t2 = st2.isEmpty() ? null: st2.peek();

    if(t1 != null && t2 != null && t1.data == t2.data &&  t1.
    {
```

see more

• Reply • Share ›

Anonymous ➚ Nagaraju • 11 months ago

Can you please tell me the concept you are applying here instead of pa

• Reply • Share ›

Alex • a year ago

Nice One

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

• Reply • Share ›

Subscribe Add Disqus to your site