Iterative Postorder Traversal | Set 2 (Using One Stack)

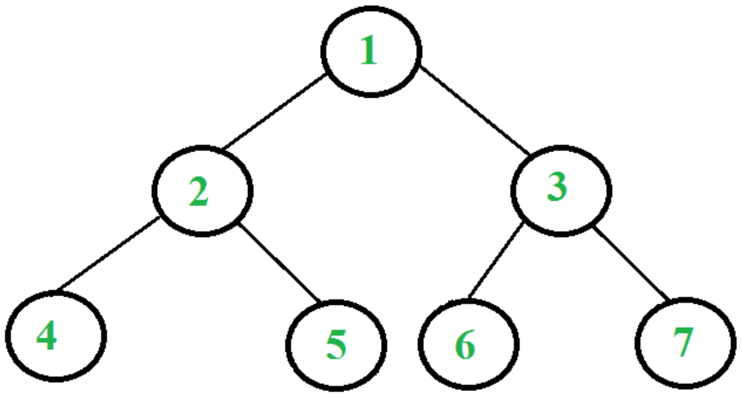
We have discussed a simple iterative postorder traversal using two stacks in the previous post. In this post, an approach with only one stack is discussed.

The idea is to move down to leftmost node using left pointer. While moving down, push root and root's right child to stack. Once we reach leftmost node, print it if it doesn't have a right child. If it has a right child, then change root so that the right child is processed before.

Following is detailed algorithm.

```
1.1 Create an empty stack
2.1 Do following while root is not NULL
    a) Push root's right child and then root to stack.
    b) Set root as root's left child.
2.2 Pop an item from stack and set it as root.
    a) If the popped item has a right child and the right child
       is at top of stack, then remove the right child from stack,
       push the root back and set root as root's right child.
    b) Else print root's data and set root as NULL.
2.3 Repeat steps 2.1 and 2.2 while stack is not empty.
```

Let us consider the following tree



Following are the steps to print postorder traversal of the above tree using one stack.

```
1. Right child of 1 exists.
   Push 3 to stack. Push 1 to stack. Move to left child.
        Stack: 3, 1

2. Right child of 2 exists.
   Push 5 to stack. Push 2 to stack. Move to left child.
        Stack: 3, 1, 5, 2

3. Right child of 4 doesn't exist. '
   Push 4 to stack. Move to left child.
        Stack: 3, 1, 5, 2, 4

4. Current node is NULL.
   Pop 4 from stack. Right child of 4 doesn't exist.
   Print 4. Set current node to NULL.
        Stack: 3, 1, 5, 2

5. Current node is NULL.
    Pop 2 from stack. Since right child of 2 equals stack top element,
    pop 5 from stack. Now push 2 to stack.
    Move current node to right child of 2 i.e. 5
        Stack: 3, 1, 2

6. Right child of 5 doesn't exist. Push 5 to stack. Move to left child.
        Stack: 3, 1, 2, 5

7. Current node is NULL. Pop 5 from stack. Right child of 5 doesn't exist.
   Print 5. Set current node to NULL.
        Stack: 3, 1, 2

8. Current node is NULL. Pop 2 from stack.
   Right child of 2 is not equal to stack top element.
   Print 2. Set current node to NULL.
        Stack: 3, 1

9. Current node is NULL. Pop 1 from stack.
   Since right child of 1 equals stack top element, pop 3 from stack.
   Now push 1 to stack. Move current node to right child of 1 i.e. 3
        Stack: 1

10. Repeat the same as above steps and Print 6, 7 and 3.
    Pop 1 and Print 1.
```

C

```
// C program for iterative postorder traversal using one stack
#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--];
}

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

// An iterative function to do postorder traversal of a given binary tree
void postOrderIterative(struct Node* root)
{
    // Check for empty tree
    if (root == NULL)
        return;

    struct Stack* stack = createStack(MAX_SIZE);
    do
    {
        // Move to leftmost node
        while (root)
        {
            // Push root's right child and then root to stack.
            if (root->right)
                push(stack, root->right);
            push(stack, root);

            // Set root as root's left child
            root = root->left;
        }

        // Pop an item from stack and set it as root
        root = pop(stack);

        // If the popped item has a right child and the right child is not
        // processed yet, then make sure right child is processed before root
        if (root->right && peek(stack) == root->right)
        {
            pop(stack);  // remove right child from stack
            push(stack, root);  // push root back to stack
            root = root->right; // change root so that the right
                                // child is processed next
        }
        else  // Else print root's data and set root as NULL
        {
            printf("%d ", root->data);
            root = NULL;
        }
    } while (!isEmpty(stack));
}

// 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);
    printf("Post order traversal of binary tree is :\n");
    printf("[");
    postOrderIterative(root);
    printf("]");


    return 0;
}
```

Java

```
// A java program for iterative postorder traversal using stack
import java.util.ArrayList;
import java.util.Stack;

// A binary tree node
class Node {

    int data;
    Node left, right;

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

class BinaryTree {

    static Node root;
    ArrayList<Integer> list = new ArrayList<Integer>();

    // An iterative function to do postorder traversal of a given binary tree
    ArrayList<Integer> postOrderIterative(Node node) {
        Stack<Node> S = new Stack<Node>();

        // Check for empty tree
        if (node == null) {
            return list;
        }
        S.push(node);
        Node prev = null;
        while (!S.isEmpty()) {
            Node current = S.peek();

            /* go down the tree in search of a leaf an if so process it and pop
            stack otherwise move down */
            if (prev == null || prev.left == current || prev.right == current) {
                if (current.left != null) {
                    S.push(current.left);
                } else if (current.right != null) {
                    S.push(current.right);
                } else {
                    S.pop();
                    list.add(current.data);
                }

                /* go up the tree from left node, if the child is right
                push it onto stack otherwise process parent and pop stack */
            } else if (current.left == prev) {
                if (current.right != null) {
                    S.push(current.right);
                } else {
                    S.pop();
                    list.add(current.data);
                }

                /* go up the tree from right node and after coming back
                 from right node process parent and pop stack */
            } else if (current.right == prev) {
                S.pop();
                list.add(current.data);
            }

            prev = current;
        }

        return list;
    }
    // Driver program to test above functions
    public static void main(String args[]) {

        BinaryTree tree = new BinaryTree();

        // Let us create trees shown in above diagram
        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);

        ArrayList<Integer> mylist = tree.postOrderIterative(root);

        System.out.println("Post order traversal of binary tree is :");
        System.out.println(mylist);

    }
}

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

Python

```
# Python program for iterative postorder traversal
# using one stack

# Stores the answer
ans = []

# 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

def peek(stack):
    if len(stack) > 0:
        return stack[-1]
    return None
# A iterative function to do postorder traversal of
# a given binary tree
```

```
def postOrderIterative(root):

    # Check for empty tree
    if root is None:
        return

    stack = []

    while(True):

        while (root):
             # Push root's right child and then root to stack
             if root.right is not None:
                stack.append(root.right)
             stack.append(root)

             # Set root as root's left child
             root = root.left

        # Pop an item from stack and set it as root
        root = stack.pop()

        # If the popped item has a right child and the
        # right child is not processed yet, then make sure
        # right child is processed before root
        if (root.right is not None and
            peek(stack) == root.right):
            stack.pop() # Remove right child from stack
            stack.append(root) # Push root back to stack
            root = root.right # change root so that the
                              # righ childis processed next

        # Else print root's data and set root as None
        else:
            ans.append(root.data)
            root = None

        if (len(stack) <= 0):
                break

# Driver pogram to test above function
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
root.right.left = Node(6)
root.right.right = Node(7)

print "Post Order traversal of binary tree is"
```

```
postOrderIterative(root)
print ans

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Output:

```
Post Order traversal of binary tree is
[4, 5, 2, 6, 7, 3, 1]
```

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

Spend less time debugging, more time coding.
Get Visual Studio Code FREE
Microsoft

60 Comments Category: Stack Trees Tags: stack

Related Posts:

- Check if a given array can represent Preorder Traversal of Binary Search Tree
- Minimum number of bracket reversals needed to make an expression balanced
- Iterative Depth First Traversal of Graph
- Sort a stack using recursion
- Length of the longest valid substring
- Find maximum of minimum for every window size in a given array
- Iterative Tower of Hanoi
- How to efficiently implement k stacks in a single array?

(Login to Rate and Mark)

4.2 Average Difficulty : **4.2/5.0**
Based on **15** vote(s)

Add to TODO List

Mark as DONE

Like Share 26 people like this.

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

Debabrata

```
//Using double stack is simpler
struct Stack{
Node *treeNode;
struct Stack *next;
};
typedef struct Stack Stack;
void push(Stack **root,Node *t_Node){
Stack *head = *root,*tmp;
tmp = NALLOC(1,Stack);
tmp->treeNode = t_Node;
tmp->next = (*root);
(*root) = tmp;
}
Node* pop(Stack **root){
Stack *tmp = *root;
Node* ret;
if(!tmp){
return NULL;
}
*root = tmp->next;
tmp->next = NULL;
ret = tmp->treeNode;
free (tmp);
return ret;
}
void it_postorder(Node *root){
Stack *S1 = NULL;
Stack *S2 = NULL ;
Node *tmp ;
push(&S1,root);
while(S1 ){
tmp = pop(&S1);
push(&S2,tmp);
if(tmp->left){
push(&S1,tmp->left);
}
if(tmp->right){
push(&S1,tmp->right);
}
}
std::cout<<"n";
while(S2){
tmp = pop(&S2);
if(!tmp){
break;
}
std::cout<data<“;
}
std::cout<<"n";
}
```

Arko

will this code work??/

```
private static void iterativePostorderSingleStack(Node root) {
if (root != null) {
```

StackArray<Node> array = new StackArray<Node>();
array.push(root);
while (!array.isEmpty()) {
Node node = array.pop();
if (node.left == null && node.right == null)
System.out.print(node.item + " ");
else {
Node left = node.left;
Node right = node.right;
node.left = null;
node.right = null;
array.push(node);
if (right != null)
array.push(right);
if (left != null)
array.push(left);
}
}
}
}

Trilok Sharma

```
/*
if (root->right && !isEmpty(stack) && peek(stack) == root->right)
should be used in place of
if (root->right && peek(stack) == root->right)
//---------------------------------------------------
// CORRECT CODE IS

void postOrderIterative(struct Node* root)
{
        // Check for empty tree
    if (root == NULL)
        return;

  stack<struct Node *> mystack;

    do
    {
        // Move to leftmost node
        while (root)
        {
            // Push root's right child and then root to stack.
            if (root->right)
                mystack.push(root->right);

            mystack.push(root);

            // Set root as root's left child
            root = root->left;
        }

        // Pop an item from stack and set it as root
        root = mystack.top();
        mystack.pop();

        // If the popped item has a right child and the right child is not
        // processed yet, then make sure right child is processed before root

        if (root->right && !mystack.empty() && mystack.top() == root->right)
        {
            mystack.pop();  // remove right child from stack
            mystack.push(root);  // push root back to stack
            root = root->right; // change root so that the right
                                // child is processed next
        }
        else  // Else print root's data and set root as NULL
        {
            printf("%d ", root->data);
            root = NULL;
        }
    } while (!mystack.empty());
}






 */
```

Trilok Sharma

```
/*

if (root->right && !isEmpty(stack) && peek(stack) == root->right)
should be used in place of
if (root->right && peek(stack) == root->right)

 */
```

Geek86
Please validate my code for iterative post order traversal..
I have used 2 stacks..
public class Client {
public static void postOrderIterate(BTNode node) {
Stack<BTNode> processingStack = new Stack<BTNode>();
Stack<BTNode> resultStack = new Stack<BTNode>();
processingStack.push(node);
while (!processingStack.isEmpty()) {
BTNode processingNode = processingStack.pop();
if (processingNode.getLeft() != null) {
processingStack.push(processingNode.getLeft());
}
if (processingNode.getRight() != null) {
processingStack.push(processingNode.getRight());
}
resultStack.add(processingNode);
}
while (!resultStack.isEmpty()) {
System.out.println("Processing " + resultStack.pop());
}
}
public static void main(String[] args) {
BTNode one = new BTNode(1);
BTNode two = new BTNode(2);
BTNode three = new BTNode(3);
BTNode four = new BTNode(4);
BTNode five = new BTNode(5);
BTNode six = new BTNode(6);
BTNode seven = new BTNode(7);
one.setLeft(two);
one.setRight(three);
two.setLeft(four);
two.setRight(five);
three.setLeft(six);
three.setRight(seven);
seven.setRight(new BTNode(8));
postOrderIterate(one);
}
}
Output:
Processing 4
Processing 5
Processing 2
Processing 6
Processing 8
Processing 7
Processing 3
Processing 1

EOF
Much simpler solution if we are allowed to destroy the tree 😃
Here is a java code.

```
    //JAVA Code
    static void traverse(Node root){
        Stack<Node> s = new Stack<>();
        Node temp;

        s.push(root);
        while(!s.empty()){
            temp = s.peek();
            if(temp.left != null){
                s.push(temp.left);
                temp.left = null;
            }else if(temp.right != null){
                s.push(temp.right);
                temp.right = null;
            }else if(temp.left == null && temp.right == null){
                System.out.print(temp.keyC + " ");
                s.pop();
            }
        }
    }
```

EOF
We can copy the tree before traversing..
EOF
?
Anonym
looks like you could have used a bool to track the visited nodes
abhishek08aug
Intelligent 😃
Sibendu Dey
#include
#include
using namespace std;
struct node {
int data;
struct node *leftchild;
struct node *rightchild;
};
struct node * create_node(int data) {
struct node *temp=(struct node *)malloc(sizeof(struct node));
temp->data=data;
temp->leftchild=temp->rightchild=NULL;
}
void inorder(struct node *root) {
struct node **stack=(struct node **)malloc(sizeof *stack * 20);
struct node *current=NULL,*temp;
int index=-1,previndex;
stack[++index]=root;
current=root;
while(1) {
if(current->leftchild!=NULL) {
index=index+1;
stack[index]=current->leftchild;
temp=current;
current=current->leftchild;
temp->leftchild=NULL;
}
else {
if(current->rightchild!=NULL) {
temp=current;
current=current->rightchild;
temp->rightchild=NULL;
index=index+1;
previndex=index-1;
stack[index]=current;
}

else {
cout<data<0) {
cout<data<leftchild=create_node(2);
root->leftchild->leftchild=create_node(4);
root->leftchild->leftchild->leftchild=create_node(5);
root->leftchild->leftchild->leftchild->rightchild=create_node(9);
root->leftchild->leftchild->leftchild->rightchild->leftchild=create_node(10);
root->leftchild->leftchild->leftchild->rightchild->rightchild=create_node(11);
root->leftchild->rightchild=create_node(6);
root->rightchild=create_node(3);
root->rightchild->leftchild=create_node(7);
root->rightchild->leftchild->rightchild=create_node(8);
inorder(root);
}
Kumar

```
// An iterative function to do postorder traversal of a given binary tree
void postOrderIterative(struct Node* root)
{
  if(root==NULL)
   return;

  struct Stack* stack = createStack(MAX_SIZE);
  push(stack,root);
  struct Node *prev, *curr;
  prev=NULL;

  while(!isEmpty(stack))
  {
    curr=top(stack);
    if(prev==NULL||prev->left==curr||prev->right==curr) //top to bottom
     {
      if(curr->left)
        push(stack,curr->left);
      else if(curr->right)
        push(stack,curr->right);
      else
       {
        printf("%d ",curr->data);
        pop(stack);
       }
     }
    else if(curr->left==prev) //bottom up
     {
       if(curr->right)
        {
          push(stack,curr->right);
        }
       else
        {
           printf("%d ",curr->data);
           pop(stack);
        }
     }
    else if(curr->right==prev)
     {
        printf("%d ",curr->data);
        pop(stack);
     }
    prev=curr;
  }
}
```