Traversing a binary tree recursively is usually the first approach towards approaching binary tree problems.

When traversing a tree iteratively it is common to use a **stack** or a **queue**. The common pattern involves: 1) Determine whether to use a stack or a queue to store nodes we need to visit a) stacks are last-in-first-out b) queues are first-in-first-out 2) While our stack/queue is not null, retrieve nodes from it a) When we retrieve a node to visit it, we also have to figure out how to put its child nodes on

As an example, we can take a look at how to implement a preorder traversal iteratively.

Recall the **recursive approach** for a preorder traversal:

**void** **printPreorder**(TreeNode node) {

**if** (node **==** null) {

**return**;

}

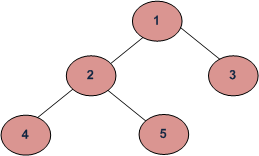
System.out.print(node.data **+** " "); *// process node*

printPreorder(node.left); *// recurse on left*

printPreorder(node.right); *// recurse on right*

}

For the following tree:



Our preorder traversal would be: 1 -> 2 -> 4 -> 5 -> 3

At this point, we know a couple of things

* We want to visit roots before leaves
* We want to visit the left child before the right child.

1) Let's say we go with the stack approach. The first node we add will always be the root, in this case 1.

2) When we pop 1 from the stack, we have the option to add node 2 first or node 3 first. Which node should we push onto the stack first?

* Looking at what our traversal should end up being, 2 comes before 3, so if we want to see 2 first, we should probably add node 3 to the stack, followed by node 2. That way, when we pop from the stack, 2 will be popped before 3

At this point, we've printed 1 and our stack looks like:

(2)

(3)

3) Now that our stack is not empty, we can pop from it again. We pop2 from it, and again we have to decide whether to push node 4 first or node 5 first.

* Again, if we look at our desired traversal outcome from our recursive approach, we see that 4 should be printed before 5. Following step 2a, it looks like we should push 5 onto the stack first, followed by 4.
* Now we've printed 1 2 and our stack looks like

(4) (5) (3)

4) Again, we pop from our stack. This time we 4 is popped and printed, and since 4 has no children, we don't add anything, and just keep popping.

* After 4 has been popped, we will have printed 1 2 4 and our stack would then contain:

(5) (3)

Looking at what we've been doing, it looks like a pattern has emerged.

1. Create empty stack
2. Push root node onto stack
3. While our stack is not empty: a. pop node from the stack and print it b. push right child of popped node to stack. c. push left child of popped node to stack

# 

# Binary Tree-postorder Traversal – Non Recursive Approach

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

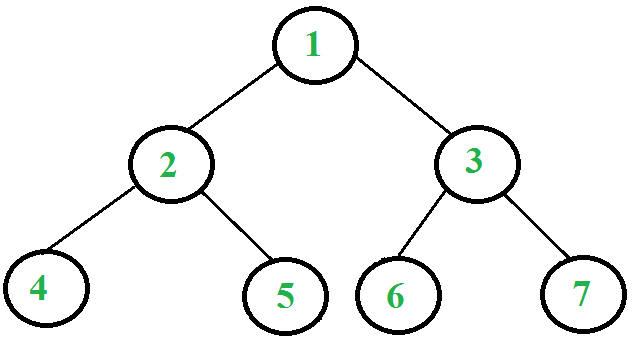
is at top of stack, then remove the right child from stack,

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

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.

Binary Tree-Inorder Traversal – Non Recursive Approach

**Pseudo Code:**

1. Create a Stack.
2. Push the root into the stack and set the root = root.left continue till it hits the NULL.
3. If root is null and Stack is empty Then
   1. return, we are done.
4. Else
   1. Pop the top Node from the Stack and set it as, root = popped\_Node.
   2. print the root and go right, root = root.right.
   3. Go to step 2.
5. End If