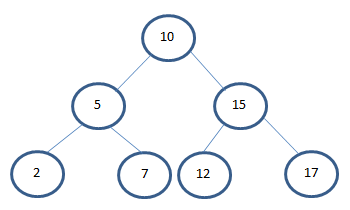
Binary Search Tree InOrder traversal using Non-Recursion.

In previous post we studied how to traverse a tree by InOrder Recursion. To see how to insert a node in tree click [here](http://data-structure-learning.blogspot.com/2015/05/part-1-binary-search-tree-introduction.html).

In this post we will study how to traverse a tree by Non-Recursion InOrder traversal.



To reiterate how to traverse a tree in InOrder traversal by recursion we follow 3 steps:

3 step description for InOrder traversal

Traverse left subtree by calling function inOrder recursively.

Display the current element.

Traverse the right subtree by calling inOrder function recursively.

We will modify the above algorithm and traverse without recursion.

So a modified algorithm will look like following:

Use Stack<E> class of Java Collection Framework to store the nodes.

Stack<Node> s = **new** Stack<Node>();

Store root in temp as we don’t want root to move anywhere.

Node temp = root;

Now check if stack is empty or temp object is null

**while** (!s.isEmpty() || temp != **null**)

Now check if temp is null or not. If temp is not null then push nodes in stack. This step is similar to step 1 of InOrder traversal using Recursion. As soon as you push temp in stack traverse to left child.

**if** (temp != **null**) {

s.push(temp);

temp = temp.leftChild;

}

If node is null then we need to display the current value. Pop the element from stack and then display it. Now traverse the right child. This step is similar to step 2 and 3 of InOrder using Recursion.

**else** {

temp = s.pop();

System.***out***.println(temp.iData);

temp = temp.rightChild;

}

Well, we are done with the InOrder traversal without recursion. Now let us write entire code together for better visibility. I will comment the code so it is easy to understand it.

/\*\*

\* InOrder Non-Recursion using a Stack

\* \*/

**public** **void** in(Node root) {

//Define a Stack to store nodes into it.

Stack<Node> s = **new** Stack<Node>();

//Assign root to temp. Do not move root.

Node temp = root;

/\*\*

\* Here we check

\* 1. Whether stack is empty or not

\* 2. temp is not null

\* \*/

**while** (!s.isEmpty() || temp != **null**) {

/\*\*

\* temp is not null

\* 1. Push temp in stack

\* 2. traverse left child

\*

\* This step is similar to Step 1 in InOrder Recursion.

\* \*/

**if** (temp != **null**) {

s.push(temp);

temp = temp.leftChild;

}

/\*\*

\* temp is null

\* 1. Push temp in stack

\* 2. traverse right child

\*

\* This step is similar to Step 2 & 3 in InOrder Recursion.

\* \*/

**else** {

temp = s.pop();

System.***out***.print(temp.iData+" ");

temp = temp.rightChild;

}

}

}

Output:

2 5 7 10 12 15 17

In next post we will see how to traverse PreOrder by Recursion and also PreOrder by Non-Recursion.