Traversing BST - Recorriendo un BST

Traversing Binary Trees

Inorder traversal prints out the nodes "in order."

There are two other major types of traversal - preorder and postorder. The difference between the traversals is the order in which the root of a subtree is visited. (A 'visit' is a function which does something at each node, typically printing it, reading some value stored at the node, or updating some value in all the nodes.)

Preorder: Visit the root of the tree (or subtree), then (recursively) traverse the left subtree, then traverse the right subtree.

Inorder: Traverse the left subtree, then visit the root, then traverse the right subtree.

Postorder: Traverse the left subtree, then traverse the right subtree, then visit the root.

Recorriendo árboles binarios

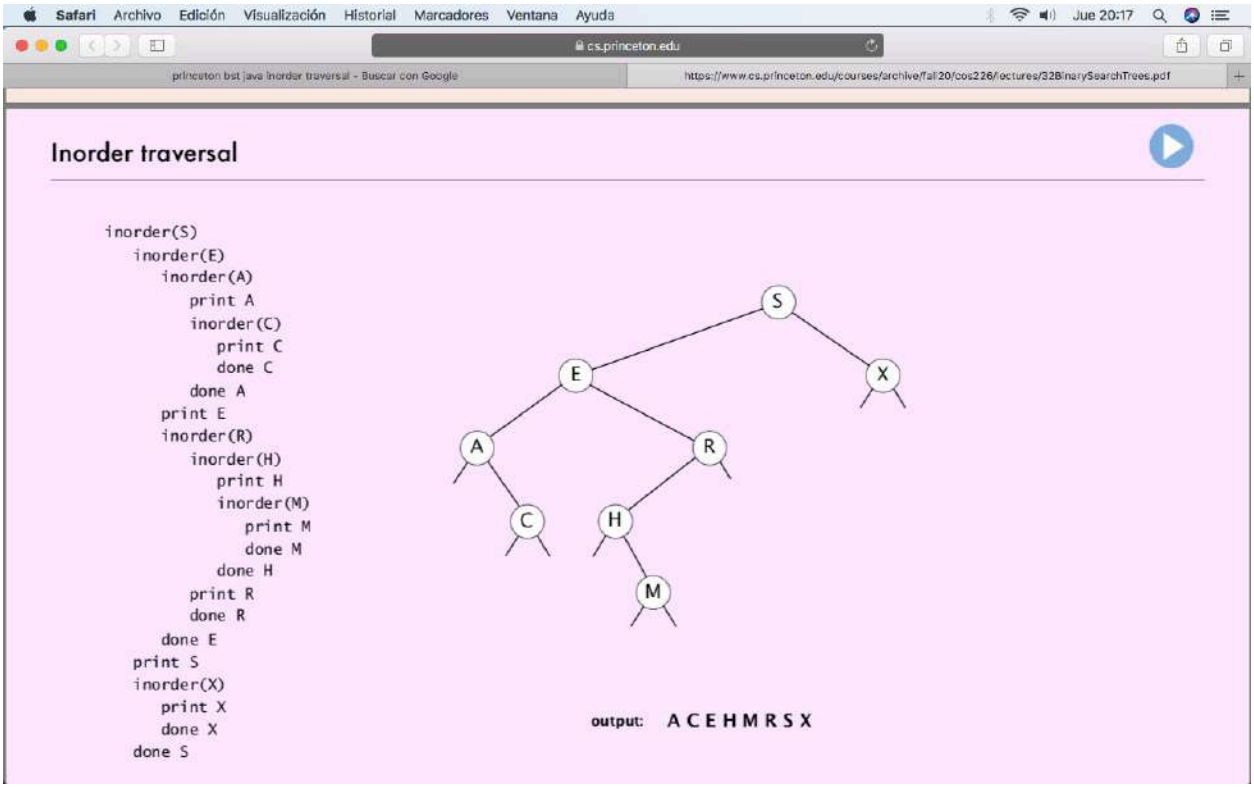
El recorrido Inorder imprime los nodos "en orden".

Hay otros dos tipos principales de recorrido: preorder y postorder. La diferencia entre los recorridos es el orden en el que se visita la raíz de un subárbol. (Una 'visita' es una función que hace algo en cada nodo, por lo general imprimir, leer algún valor almacenado en el nodo o actualizar algún valor en todos los nodos).

Preorder: Visita la raíz del árbol (o subárbol), luego (recursivamente) recorre el subárbol izquierdo, luego recorre el subárbol derecho.

Inorder: Recorre el subárbol izquierdo, luego visita la raíz, luego recorre el subárbol derecho.

Postorder: Recorre el subárbol izquierdo, luego recorre el subárbol derecho, luego visita la raíz.



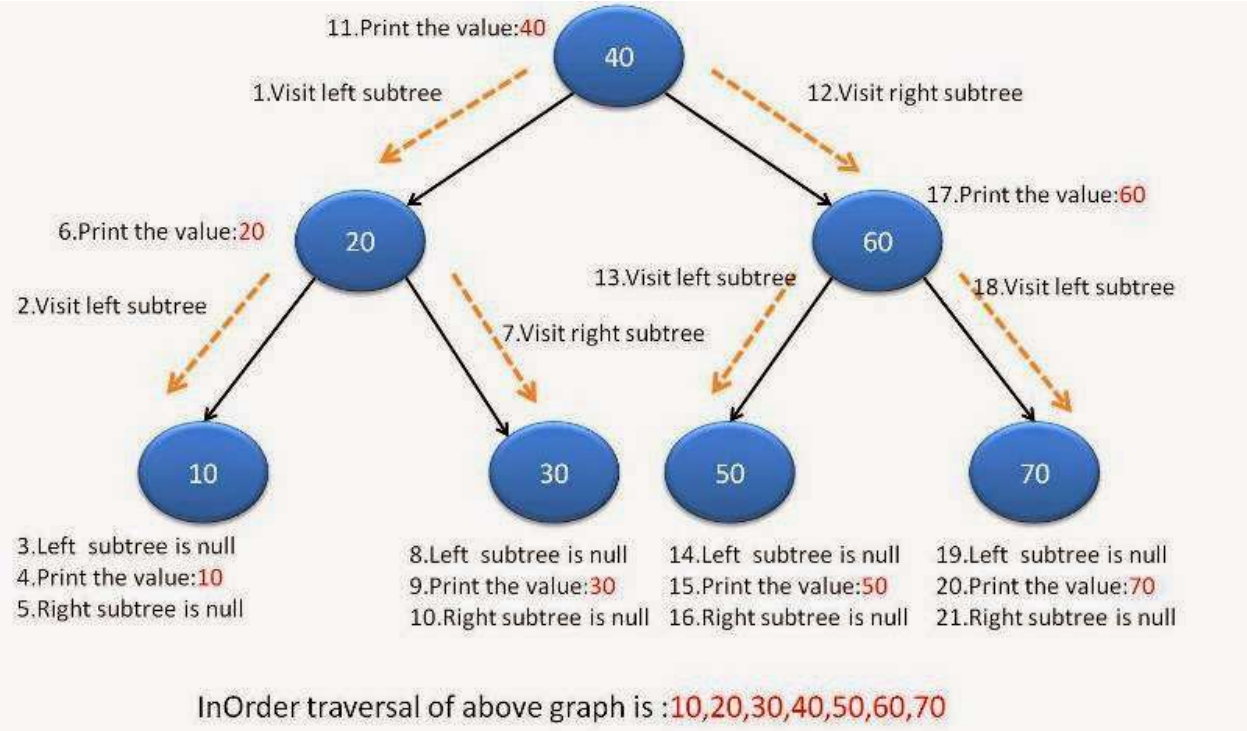
InOrder traversal:

In InOrder traversal,each node is processed between subtrees. In simpler words, visit left subtree, node and then right subtree.

Steps for InOrder traversal are:

- Traverse the left subtree in InOrder.
- Visit the node.
- Traverse the right subtree in InOrder.

Recursive solution



Code for recursion will be:

```
// Recursive Solution
      public void inOrder(TreeNode root) {
            if(root !=  null) {
                  inOrder(root.left);
                  //Visit the node by Printing the node data
                  System.out.printf("%d ",root.data);
                  inOrder(root.right);
            }
      }
```

Iterative solution

For recursion, we use implicit stack. So here to convert recursive solution to iterative, we will use explicit stack.

Steps for iterative solution:

- Create an empty stack s and Initialize currentNode as root
- Push the currentNode to s and set currentNode = currentNode.left until currentNode is NULL
- If currentNode is NULL and s is not empty then
  - Pop the top node from stack s and print it
  - set currentNode = currentNode.right
  - go to step 2
- If stack is empty and currentNode is also null then we are done with it.

```
// Iterative solution
	public void inOrderIter(TreeNode root) {

		if(root == null)
			return;

		Stack<TreeNode> s = new Stack<TreeNode>();
		TreeNode currentNode=root;

		while(!s.empty() || currentNode!=null){

			if(currentNode!=null)
			{
				s.push(currentNode);
				currentNode=currentNode.left;
			}
			else
			{
				TreeNode n=s.pop();
				System.out.printf("%d ",n.data);
				currentNode=n.right;
			}
		}
	}
```

Lets create java program for InOrder traversal:

```
import java.util.Stack;

public class BinaryTreeInOrder {

	public static class TreeNode
	{
		int data;
		TreeNode left;
		TreeNode right;
		TreeNode(int data)
		{
			this.data=data;
		}
	}

	// Recursive Solution
```

```
	public void inOrder(TreeNode root) {
		if(root !=  null) {
			inOrder(root.left);
			//Visit the node by Printing the node data
			System.out.printf("%d ",root.data);
			inOrder(root.right);
		}
	}

	// Iterative solution
	public void inOrderIter(TreeNode root) {

		if(root == null)
			return;

		Stack<TreeNode> s = new Stack<TreeNode>();
		TreeNode currentNode=root;

		while(!s.empty() || currentNode!=null){

			if(currentNode!=null)
			{
				s.push(currentNode);
				currentNode=currentNode.left;
			}
			else
			{
				TreeNode n=s.pop();
				System.out.printf("%d ",n.data);
				currentNode=n.right;
			}
		}
	}

	public static void main(String[] args)
	{
		BinaryTreeInOrder bi=new BinaryTreeInOrder();
		// Creating a binary tree
		TreeNode rootNode=createBinaryTree();
		System.out.println("Using Recursive solution:");

		bi.inOrder(rootNode);

		System.out.println();
		System.out.println("-------------------------");
		System.out.println("Using Iterative solution:");

		bi.inOrderIter(rootNode);
	}

	public static TreeNode createBinaryTree()
	{

		TreeNode rootNode =new TreeNode(40);
```

```
            TreeNode node20=new TreeNode(20);
            TreeNode node10=new TreeNode(10);
            TreeNode node30=new TreeNode(30);
            TreeNode node60=new TreeNode(60);
            TreeNode node50=new TreeNode(50);
            TreeNode node70=new TreeNode(70);

            rootNode.left=node20;
            rootNode.right=node60;

            node20.left=node10;
            node20.right=node30;

            node60.left=node50;
            node60.right=node70;

            return rootNode;
        }
}
```

Run above program and you will get following output:

```
Using Recursive solution:
10 20 30 40 50 60 70
_________-
Using Iterative solution:
10 20 30 40 50 60 70
```

PreOrder traversal:

In PreOrder traversal,each node is processed before either of its sub-trees.In simpler words,Visit each node before its children.

Steps for PreOrder traversal are:

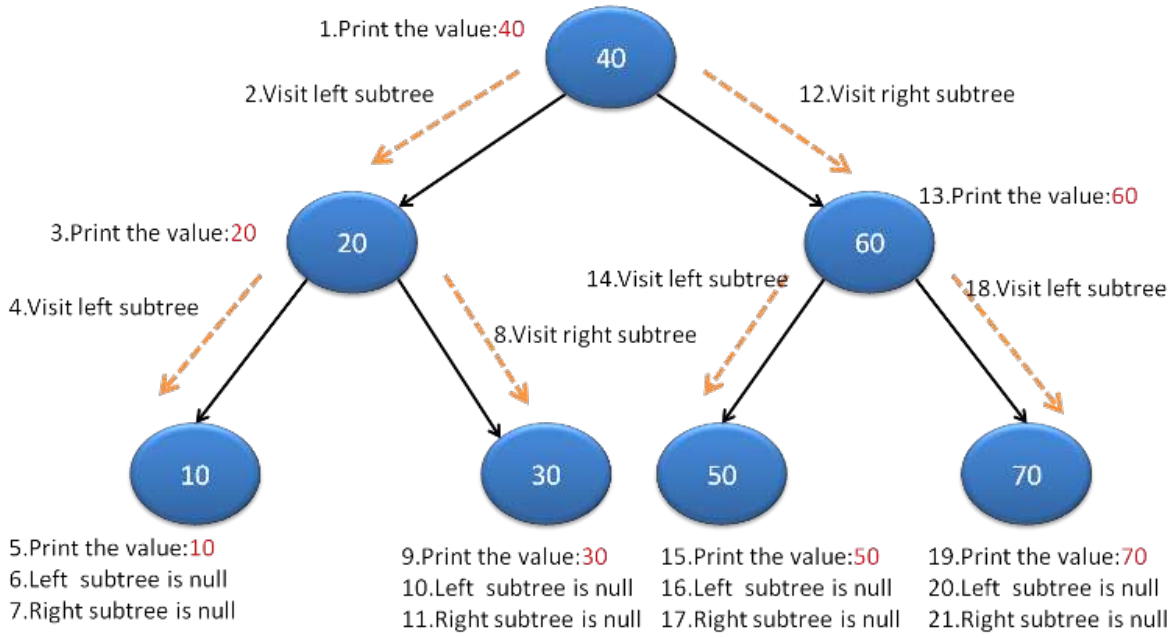
- Visit the node.
- Traverse the left subtree in PreOrder.
- Traverse the right subtree in PreOrder.

There can be two ways of implementing it

- Recursive
- Iterative

Recursive solution:

Recursive solution is very straight forward.Below diagram will make you understand recursion better.



PreOrder traversal of above graph is :40,20,10,30,60,50,70

Code for recursion will be:

```
public void preorder(TreeNode root) {
    if(root != null) {
   //Visit the node by Printing the node data
      System.out.printf("%d ",root.data);
      preorder(root.left);
      preorder(root.right);
    }
  }
```

Iterative solution:

For recursion, we use implicit stack. So here to convert recursive solution to iterative, we will use explicit stack.

Steps for iterative solution:

- Create empty stack and pust root node to it.
- Do the following when stack is not empty
  - Pop a node from stack and print it
  - Push right child of popped node to stack
  - Push left child of popped node to stack

We are pushing right child first, so it will be processed after left subtree as Stack is LIFO.

```
public void preorderIter(TreeNode root) {

        if(root == null)
            return;

        Stack<TreeNode> stack = new Stack<TreeNode>();
        stack.push(root);

        while(!stack.empty()){

            TreeNode n = stack.pop();
            System.out.printf("%d ",n.data);

            if(n.right != null){
                stack.push(n.right);
            }
            if(n.left != null){
                stack.push(n.left);
            }

        }

    }
```

Run above program and you will get following output:

```
Using Recursive solution:
40 20 10 30 60 50 70
---------
Using Iterative solution:
40 20 10 30 60 50 70
```

PostOrder traversal

In PostOrder traversal, each node is processed after subtrees traversal.In simpler words,Visit left subtree, right subtree and then node.
Steps for PostOrder traversal are:

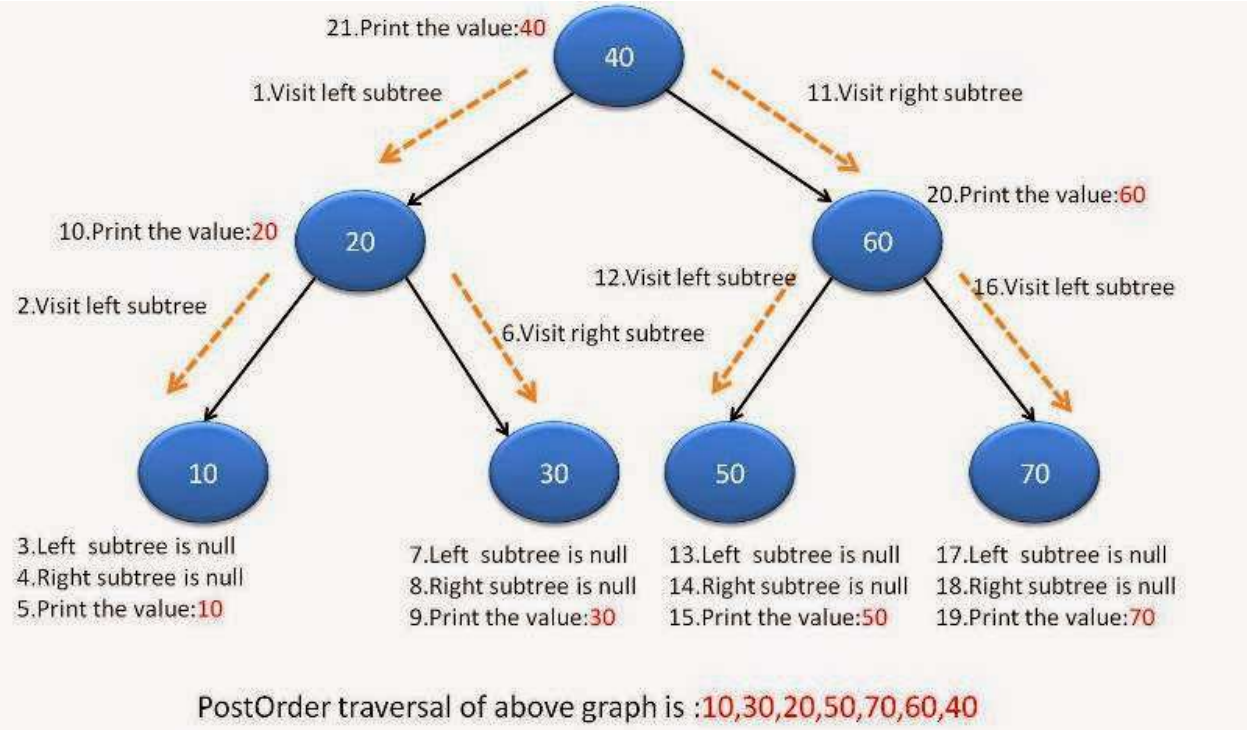
- Traverse the left subtree in PostOrder.
- Traverse the right subtree in PostOrder.
- Visit the node.

There can be two ways of implementing it

- Recursive
- Iterative

Recursive solution

Recursive solution is very straight forward.Below diagram will make you understand recursion better.



Code for recursion will be:

```
// Recursive Solution
      public void postOrder(TreeNode root) {
            if(root !=  null) {
                  postOrder(root.left);
                  postOrder(root.right);
                  //Visit the node by Printing the node data
                  System.out.printf("%d ",root.data);
            }
      }
```

Iterative solution:

Steps for iterative solution:

- Create an empty stack s and set currentNode =root.
- while currentNode is not NULL Do following
  - Push currentNode 's right child and then currentNode to stack s
  - Set currentNode=currentNode.left
- Pop a node from stack s and set it to currentNode
  - If the popped node has a right child and the right child is at top of stack, then remove the right child from stack, push the currentNode back and set currentNode as currentNode 's right child.
  - Else print currentNode's data and set currentNode as NULL.
- Repeat steps 2 and 3 while stack is not empty.

```
// Iterative solution
      public void postorderIter( TreeNode root) {
            if( root == null ) return;

            Stack<TreeNode> s = new Stack<TreeNode>( );
            TreeNode current = root;

            while( true ) {

                  if( current != null ) {
                        if( current.right != null )
                              s.push( current.right );
                        s.push( current );
                        current = current.left;
                        continue;
                  }

                  if( s.isEmpty( ) )
                        return;
                  current = s.pop( );

                  if( current.right != null && ! s.isEmpty( ) &&
                        current.right == s.peek( ) ) {
                        s.pop( );
                        s.push( current );
                        current = current.right;
                  } else {
                        System.out.print( current.data + " " );
                        current = null;
                  }
            }
      }
```

Run above program and you will get following output:

```
Using Recursive solution:
10 30 20 50 70 60 40
---------
Using Iterative solution:
10 30 20 50 70 60 40
```