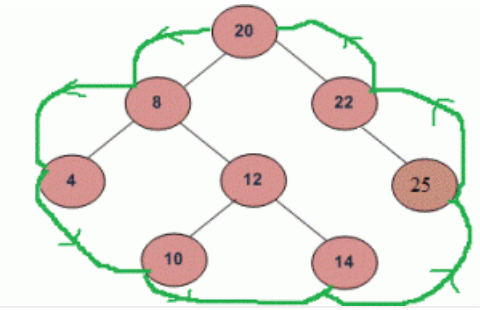
Boundary Traversal of binary tree

Given a binary tree, print boundary nodes of the binary tree Anti-Clockwise starting from the root. For example, boundary traversal of the following tree is “20 8 4 10 14 25 22"



We break the problem in 3 parts:

1. Print the left boundary in top-down manner.

2. Print all leaf nodes from left to right, which can again be sub-divided into two sub-parts:

.....**2.1** Print all leaf nodes of left sub-tree from left to right.

.....**2.2** Print all leaf nodes of right subtree from left to right.

3. Print the right boundary in bottom-up manner.

We need to take care of one thing that nodes are not printed again. e.g. The left most node is also the leaf node of the tree.

Based on the above cases, below is the implementation:

C++

```
/* program for boundary traversal of a binary tree */
#include <stdio.h>
#include <stdlib.h>

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct node
{
    int data;
    struct node *left, *right;
};

// A simple function to print leaf nodes of a binary tree
void printLeaves(struct node* root)
{
    if ( root )
    {
        printLeaves(root->left);

        // Print it if it is a leaf node
        if ( !(root->left)  &&  !(root->right) )
            printf("%d ", root->data);

        printLeaves(root->right);
    }
}

// A function to print all left boundry nodes, except a leaf node.
// Print the nodes in TOP DOWN manner
void printBoundaryLeft(struct node* root)
{
    if (root)
```

```
    {
        if (root->left)
        {
            // to ensure top down order, print the node
            // before calling itself for left subtree
            printf("%d ", root->data);
            printBoundaryLeft(root->left);
        }
        else if( root->right )
        {
            printf("%d ", root->data);
            printBoundaryLeft(root->right);
        }
        // do nothing if it is a leaf node, this way we avoid
        // duplicates in output
    }
}

// A function to print all right boundry nodes, except a leaf node
// Print the nodes in BOTTOM UP manner
void printBoundaryRight(struct node* root)
{
    if (root)
    {
        if ( root->right )
        {
            // to ensure bottom up order, first call for right
            //  subtree, then print this node
            printBoundaryRight(root->right);
            printf("%d ", root->data);
        }
        else if ( root->left )
        {
            printBoundaryRight(root->left);
            printf("%d ", root->data);
        }
       // do nothing if it is a leaf node, this way we avoid
       // duplicates in output
    }
}


// A function to do boundary traversal of a given binary tree
void printBoundary (struct node* root)
{
    if (root)
    {
        printf("%d ",root->data);

        // Print the left boundary in top-down manner.
        printBoundaryLeft(root->left);

        // Print all leaf nodes
        printLeaves(root->left);
        printLeaves(root->right);

        // Print the right boundary in bottom-up manner
        printBoundaryRight(root->right);
    }
}

// A utility function to create a node
struct node* newNode( int data )
{
    struct node* temp = (struct node *) malloc( sizeof(struct node) );

    temp->data = data;
    temp->left = temp->right = NULL;

    return temp;
}

// Driver program to test above functions
```

```
int main()
{
    // Let us construct the tree given in the above diagram
    struct node *root         = newNode(20);
    root->left                = newNode(8);
    root->left->left          = newNode(4);
    root->left->right         = newNode(12);
    root->left->right->left   = newNode(10);
    root->left->right->right  = newNode(14);
    root->right               = newNode(22);
    root->right->right        = newNode(25);

    printBoundary( root );

    return 0;
}
```

Java

```
//Java program to print boundary traversal of binary tree

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
class Node
{
    int data;
    Node left, right;

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

class BinaryTree
{
    Node root;

    // A simple function to print leaf nodes of a binary tree
    void printLeaves(Node node)
    {
        if (node != null)
        {
            printLeaves(node.left);

            // Print it if it is a leaf node
            if (node.left == null && node.right == null)
                System.out.print(node.data + " ");
            printLeaves(node.right);
        }
    }

    // A function to print all left boundry nodes, except a leaf node.
    // Print the nodes in TOP DOWN manner
    void printBoundaryLeft(Node node)
    {
        if (node != null)
        {
            if (node.left != null)
            {

                // to ensure top down order, print the node
                // before calling itself for left subtree
                System.out.print(node.data + " ");
                printBoundaryLeft(node.left);
            }
            else if (node.right != null)
            {
                System.out.print(node.data + " ");
```

```
                printBoundaryLeft(node.right);
            }

            // do nothing if it is a leaf node, this way we avoid
            // duplicates in output
        }
    }

    // A function to print all right boundry nodes, except a leaf node
    // Print the nodes in BOTTOM UP manner
    void printBoundaryRight(Node node)
    {
        if (node != null)
        {
            if (node.right != null)
            {
                // to ensure bottom up order, first call for right
                //  subtree, then print this node
                printBoundaryRight(node.right);
                System.out.print(node.data + " ");
            }
            else if (node.left != null)
            {
                printBoundaryRight(node.left);
                System.out.print(node.data + " ");
            }
            // do nothing if it is a leaf node, this way we avoid
            // duplicates in output
        }
    }

    // A function to do boundary traversal of a given binary tree
    void printBoundary(Node node)
    {
        if (node != null)
        {
            System.out.print(node.data + " ");

            // Print the left boundary in top-down manner.
            printBoundaryLeft(node.left);

            // Print all leaf nodes
            printLeaves(node.left);
            printLeaves(node.right);

            // Print the right boundary in bottom-up manner
            printBoundaryRight(node.right);
        }
    }

    // Driver program to test above functions
    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(20);
        tree.root.left = new Node(8);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(12);
        tree.root.left.right.left = new Node(10);
        tree.root.left.right.right = new Node(14);
        tree.root.right = new Node(22);
        tree.root.right.right = new Node(25);
        tree.printBoundary(tree.root);

    }
}

// This code has been contributed by Mayank Jaiswal(mayank_24)
```

Python

```
# Python program for binary traversal of binary tree

# 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

# A simple function to print leaf nodes of a Binary Tree
def printLeaves(root):
    if(root):
        printLeaves(root.left)

        # Print it if it is a leaf node
        if root.left is None and root.right is None:
            print root.data,

        printLeaves(root.right)

# A function to print all left boundary nodes, except a
# leaf node. Print the nodes in TOP DOWN manner
def printBoundaryLeft(root):

    if(root):
        if (root.left):

            # to ensure top down order, print the node
            # before calling itself for left subtree
            print root.data,
            printBoundaryLeft(root.left)

        elif(root.right):
            print root.data,
            printBoundaryRight(root.right)

        # do nothing if it is a leaf node, this way we
        # avoid duplicates in output


# A function to print all right boundary nodes, except
# a leaf node. Print the nodes in BOTTOM UP manner
def printBoundaryRight(root):

    if(root):
        if (root.right):
            # to ensure bottom up order, first call for
            # right subtree, then print this node
            printBoundaryRight(root.right)
            print root.data,

        elif(root.left):
            printBoundaryRight(root.left)
            print root.data,

        # do nothing if it is a leaf node, this way we
        # avoid duplicates in output


# A function to do boundary traversal of a given binary tree
def printBoundary(root):
    if (root):
        print root.data,

        # Print the left boundary in top-down manner
        printBoundaryLeft(root.left)

        # Print all leaf nodes
        printLeaves(root.left)
```

```
        printLeaves(root.right)

        # Print the right boundary in bottom-up manner
        printBoundaryRight(root.right)


# Driver program to test above function
root = Node(20)
root.left = Node(8)
root.left.left = Node(4)
root.left.right = Node(12)
root.left.right.left =  Node(10)
root.left.right.right = Node(14)
root.right = Node(22)
root.right.right = Node(25)
printBoundary(root)

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

Output:

```
20 8 4 10 14 25 22
```

Time Complexity: O(n) where n is the number of nodes in binary tree.