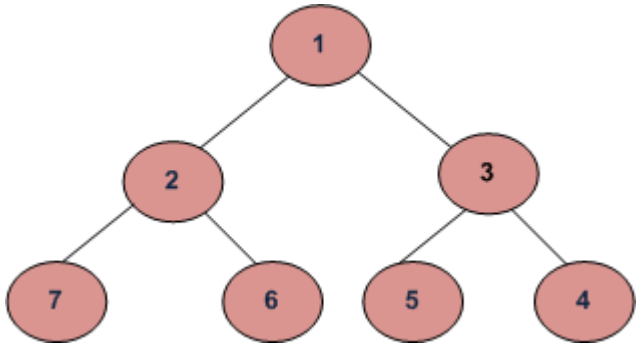
Dash
All
Articles
Videos
Problems
Quiz
Prev
Next

Tree Traversal in Spiral Form

Write a function to print spiral order traversal of a tree. For below tree, function should print 1, 2, 3, 4, 5, 6, 7.



Method 1 (Recursive)

To print the nodes in spiral order, nodes at different levels should be printed in alternating order. An additional Boolean variable *ltr* is used to change printing order of levels. If *ltr* is 1 then printGivenLevel() prints nodes from left to right else from right to left. Value of *ltr* is flipped in each iteration to change the order.

Function to print level order traversal of tree

```
printSpiral(tree)
  bool ltr = 0;
  for d = 1 to height(tree)
```

```
    printGivenLevel(tree, d, ltr);
    ltr ~= ltr /*flip ltr*/
```


Function to print all nodes at a given level

```
printGivenLevel(tree, level, ltr)
if tree is NULL then return;
if level is 1, then
    print(tree->data);
else if level greater than 1, then
    if(ltr)
        printGivenLevel(tree->left, level-1, ltr);
        printGivenLevel(tree->right, level-1, ltr);
    else
        printGivenLevel(tree->right, level-1, ltr);
        printGivenLevel(tree->left, level-1, ltr);
```

Following is the implementation of above algorithm.

C++ Java

```
// Java program for recursive level order traversal in spiral form

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

    public Node(int d)
    {
```

```
        data = d;
        left = right = null;
    }
}

class BinaryTree {
    Node root;

    // Function to print the spiral traversal of tree
    void printSpiral(Node node)
    {
        int h = height(node);
        int i;

        /* ltr -> left to right. If this variable is set then the
           given label is traversed from left to right */
        boolean ltr = false;
        for (i = 1; i <= h; i++) {
            printGivenLevel(node, i, ltr);

            /*Revert ltr to traverse next level in opposite order*/
            ltr = !ltr;
        }
    }

    /* Compute the "height" of a tree -- the number of
    nodes along the longest path from the root node
    down to the farthest leaf node.*/
    int height(Node node)
    {
        if (node == null)
            return 0;
        else {

            /* compute the height of each subtree */
            int lheight = height(node.left);
            int rheight = height(node.right);

            /* use the larger one */
            if (lheight > rheight)
                return (lheight + 1);
```

```
                else
                    return (rheight + 1);
            }
        }

        /* Print nodes at a given level */
        void printGivenLevel(Node node, int level, boolean ltr)
        {
            if (node == null)
                return;
            if (level == 1)
                System.out.print(node.data + " ");
            else if (level > 1) {
                if (ltr != false) {
                    printGivenLevel(node.left, level - 1, ltr);
                    printGivenLevel(node.right, level - 1, ltr);
                }
                else {
                    printGivenLevel(node.right, level - 1, ltr);
                    printGivenLevel(node.left, level - 1, ltr);
                }
            }
        }
        /* Driver program to test the above functions */
        public static void main(String[] args)
        {
            BinaryTree tree = new BinaryTree();
            tree.root = new Node(1);
            tree.root.left = new Node(2);
            tree.root.right = new Node(3);
            tree.root.left.left = new Node(7);
            tree.root.left.right = new Node(6);
            tree.root.right.left = new Node(5);
            tree.root.right.right = new Node(4);
            System.out.println("Spiral order traversal of Binary Tree is ");
            tree.printSpiral(tree.root);
        }
    }

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

Output

```
Spiral Order traversal of binary tree is
1 2 3 4 5 6 7
```

Time Complexity: $O(n^2)$. Worst case occurs in case of skewed trees.

Auxiliary Space: O(n) for call stack since using recursion

Method 2 (Iterative)

We can print spiral order traversal in **O(n) time** and O(n) extra space. The idea is to use two stacks. We can use one stack for printing from left to right and other stack for printing from right to left. In every iteration, we have nodes of one level in one of the stacks. We print the nodes, and push nodes of next level in other stack.

C++ Java

```java
// Java implementation of an O(n) approach of level order
// traversal in spiral form

import java.util.*;

// A Binary Tree node
class Node {
    int data;
    Node left, right;

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

```
}

class BinaryTree {

    static Node root;

    void printSpiral(Node node)
    {
        if (node == null)
            return; // NULL check

        // Create two stacks to store alternate levels
        // For levels to be printed from right to left
        Stack<Node> s1 = new Stack<Node>();
        // For levels to be printed from left to right
        Stack<Node> s2 = new Stack<Node>();

        // Push first level to first stack 's1'
        s1.push(node);

        // Keep printing while any of the stacks has some nodes
        while (!s1.empty() || !s2.empty()) {
            // Print nodes of current level from s1 and push nodes of
            // next level to s2
            while (!s1.empty()) {
                Node temp = s1.peek();
                s1.pop();
                System.out.print(temp.data + " ");

                // Note that is right is pushed before left
                if (temp.right != null)
                    s2.push(temp.right);

                if (temp.left != null)
                    s2.push(temp.left);
            }

            // Print nodes of current level from s2 and push nodes of
            // next level to s1
            while (!s2.empty()) {
                Node temp = s2.peek();
```

```
                    s2.pop();
                    System.out.print(temp.data + " ");

                    // Note that is left is pushed before right
                    if (temp.left != null)
                        s1.push(temp.left);
                    if (temp.right != null)
                        s1.push(temp.right);
                }
            }
        }

        public static void main(String[] args)
        {
            BinaryTree tree = new BinaryTree();
            tree.root = new Node(1);
            tree.root.left = new Node(2);
            tree.root.right = new Node(3);
            tree.root.left.left = new Node(7);
            tree.root.left.right = new Node(6);
            tree.root.right.left = new Node(5);
            tree.root.right.right = new Node(4);
            System.out.println("Spiral Order traversal of Binary Tree is ");
            tree.printSpiral(root);
        }
    }

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

Output

```
Spiral Order traversal of binary tree is
1 2 3 4 5 6 7
```

Time Complexity: O(n)

Auxiliary Space: O(n)

Method 3(Iterative, using Doubly Ended Queue)

The idea is to use a deque. While travelling left to right we can poll and print the elements from the front and insert their children(**left child first followed by the right child**) at the back. While travelling right to left we can poll and print the elements from the back and insert their children(**right child first followed by the left child**) at the front of the deque.

C++ Java

```
/*package whatever //do not write package name here */

import java.io.*;
import java.util.ArrayDeque;
import java.util.Deque;

class GFG {

  //Defining Node class
  static class Node {

    int key;
    Node left;
    Node right;

    public Node(int key) {
        this.key = key;
    }
}

  //Class to construct the tree
  static class MyTree {

    public MyTree(){};
```

```
    public Node root;

}

  //Function that prints the tree in spiral fashion
  public static void spiralPrint(Node root){

        //Declare a deque
        Deque<Node> dq = new ArrayDeque<>();

        //Insert the root of the tree into the deque
        dq.offer(root);

        //Create a  variable that will switch in each iteration
        boolean reverse = true;

        //Start iteration
        while (!dq.isEmpty()){

            //Save the size of the deque here itself, as in further steps the size
            //of deque will frequently change
          int n = dq.size();

            //If we are printing left to right
          if(!reverse){

            //Iterate from left to right
              for (int i =0; i < n; i++){

                //Insert the child from the back of the deque
                //Left child first
                  if (dq.peekFirst().left  != null)
                      dq.offerLast(dq.peekFirst().left);

                  if (dq.peekFirst().right != null)
                      dq.offerLast(dq.peekFirst().right);

                //Print the current processed element
                  System.out.print(dq.pollFirst().key + "  ");
```

```
                }
                //Switch reverse for next traversal
                reverse = !reverse;

            }else{

              //If we are printing right to left
              //Iterate the deque in reverse order and insert the children
              //from the front
                while (n-- >0){
                    //Insert the child in the front of the deque
                    //Right child first
                    if (dq.peekLast().right != null)
                        dq.offerFirst(dq.peekLast().right);

                    if (dq.peekLast().left != null)
                        dq.offerFirst(dq.peekLast().left);

                  //Print the current processed element
```

```
                //Switch reverse for next traversal
                reverse = !reverse;

            }
        }
    }
```

```
    public static void main (String[] args) {
        MyTree mt = new MyTree();
        mt.root = new Node(1);
        mt.root.left = new Node(2);
        mt.root.right = new Node(3);
        mt.root.left.left = new Node(7);
        mt.root.left.right = new Node(6);
        mt.root.right.left = new Node(5);
        mt.root.right.right = new Node(4);
```

```
            System.out.println("Spiral Order Traversal Of The Tree is :");
              spiralPrint(mt.root);
        }
    }

    //This code has been contributed by Abhishek Kumar Sah(kumarabhisheksah98)
```


Output

```
Spiral Order Traversal Of The Tree Is :
1  2  3  4  5  6  7
```

Time Complexity: O(N)
Auxiliary Space: O(N)