

<https://course.acciojob.com/idle?question=23641143-b9cd-45e0-b16b-80b9db53ca42>

**MEDIUM**

**Max Score: 40 Points**

## Binary Tree Level Order Traversal II

Given the `root` of a binary tree, return *the bottom-up level order traversal of its nodes' values*. (i.e., from left to right, level by level from leaf to root).

*You dont need to read input or print anything. Complete the function `reverseLevelOrder()` which takes the root of the tree as input parameter and returns a list containing the reverse level order traversal of the given tree.*

### Input Format

First line contains a string representing the tree with `root`.

The values in the string are in the order of level order traversal of the tree where, numbers denote node values, and a character "N" denotes NULL child.

### Output Format

Print the bottom-up level order traversal of the tree.

### Example 1

Input

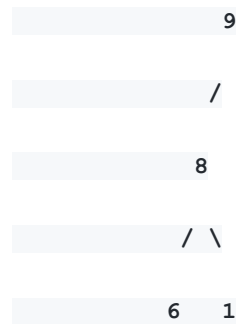
9 8 N 6 1 N N N N

Output

6 1 8 9

Explanation

The tree can be represented as:-



Level 2 from left to right is [6,1]

Level 1 from left to right is [8]

Level 0 from leaf to right is [9]

Hence the bottom-up level order traversal is [6,1,8,9]

## Example 2

Input

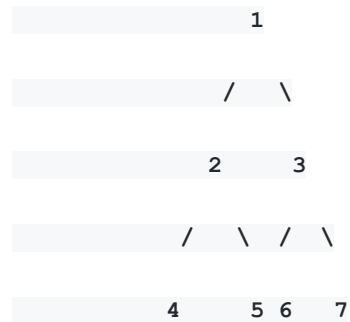
1 2 3 4 5 6 7 N N N N N N N N

Output

4 5 6 7 2 3 1

Explanation

The first tree can be represented as:-



Level 2 from left to right is [4,5,6,7]

Level 1 from left to right is [2,3]

Level 0 from leaf to right is [1]

Hence the bottom-up level order traversal is [4,5,6,7,2,3,1]

## Constraints

The number of nodes in the both the trees are in the range [1, 5000]

### Topic Tags

Trees

BFS

# My code

// in java

```
import java.util.LinkedList;
import java.util.Queue;
import java.io.*;
import java.util.*;
```

```
class Node{
    int data;
    Node left;
    Node right;
    Node(int data){
        this.data = data;
        left=null;
        right=null;
    }
}
```

```
class Main {
    static Node buildTree(String str){
        // System.out.print(str);
        if(str.length()==0 || str.charAt(0)=='N'){
            return null;
        }
        String ip[] = str.split(" ");
        Node root = new Node(Integer.parseInt(ip[0]));
```

```

Queue<Node> queue = new LinkedList<>();
queue.add(root);
int i = 1;
while(queue.size()>0 && i < ip.length) {
    Node currNode = queue.peek();
    queue.remove();
    String currVal = ip[i];
    if(!currVal.equals("N")) {
        currNode.left = new Node(Integer.parseInt(currVal));
        queue.add(currNode.left);
    }
    i++;
    if(i >= ip.length)
        break;
    currVal = ip[i];
    if(!currVal.equals("N")) {
        currNode.right = new Node(Integer.parseInt(currVal));
        queue.add(currNode.right);
    }
    i++;
}
return root;
}

void inOrder(Node node) {
    if (node == null) {
        return;
    }
    inOrder(node.left);
    System.out.print(node.data + " ");
    inOrder(node.right);
}

```

```

    }

    public static void main (String[] args) throws IOException{
        //BufferedReader br = new BufferedReader(new
        InputStreamReader(System.in));
        Scanner sc = new Scanner(System.in);

        String s = sc.nextLine();
        //String s1 = sc.nextLine();

        Node root1 = buildTree(s);
        //Node root2 = buildTree(s1);
        Solution tree = new Solution();
        ArrayList<Integer> ans =
        tree.reverseLevelOrder(root1);
        for(int i = 0 ; i < ans.size() ; ++i){
            System.out.print(ans.get(i)+" ");
        }

    }
}

```

```

class Solution{

    public ArrayList<Integer> reverseLevelOrder(Node root)
    {
        //Write your code here
    }
}

```

```

        ArrayList<Integer>al=new ArrayList<Integer>();
        Queue<Node>q=new LinkedList<>();
q.add(root);
while(q.size()>0)
{

    int n=q.size();
    for(int i=0;i<n;i++)
    {
        Node t=q.remove();
        if(t.right!=null)
        {
            q.add(t.right);
        }

        if(t.left!=null)
        {
            q.add(t.left);
        }

        al.add(t.data);

    }

}

Collections.reverse(al);
return al;
}
}

```