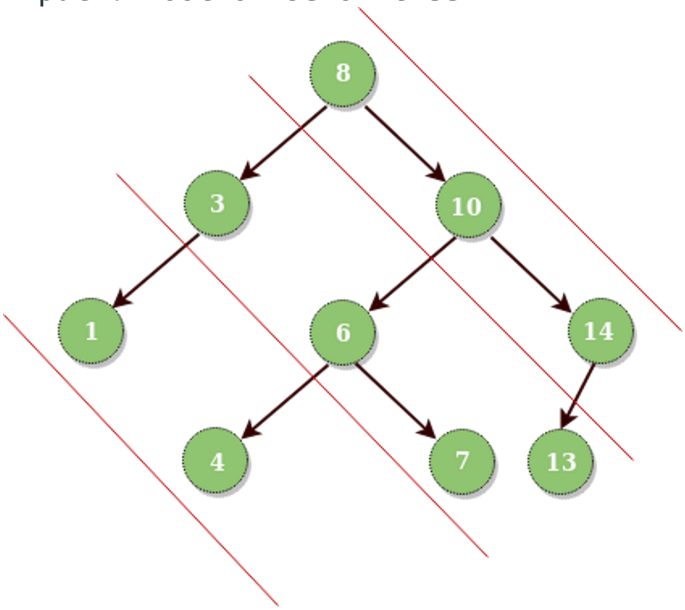
Sunday, February 6, 2022 9:27 AM

Diagonal Traversal of Binary Tree

- Difficulty Level : Medium
- Last Updated : 21 Jan, 2022

Consider lines of slope -1 passing between nodes. Given a Binary Tree, print all diagonal elements in a binary tree belonging to the same line.

```
Input : Root of below tree
```




```
Output :
Diagonal Traversal of binary tree :
 8 10 14
 3 6 7 13
 1 4
```

Observation : root and root->right values will be prioritized over all root->left values.

```
// { Driver Code Starts
//Initial Template for Java
//Contributed by Sudarshan Sharma
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 GfG {

    static Node buildTree(String str){

        if(str.length()==0 || str.charAt(0)=='N'){
            return null;
        }

        String ip[] = str.split(" ");
        // Create the root of the tree
        Node root = new Node(Integer.parseInt(ip[0]));
        // Push the root to the queue

        Queue<Node> queue = new LinkedList<>();

        queue.add(root);
        // Starting from the second element

        int i = 1;
        while(queue.size()>0 && i < ip.length) {

            // Get and remove the front of the queue
            Node currNode = queue.peek();
            queue.remove();

            // Get the current node's value from the string
            String currVal = ip[i];

            // If the left child is not null
            if(!currVal.equals("N")) {

                // Create the left child for the current node
                currNode.left = new Node(Integer.parseInt(currVal));
                // Push it to the queue
                queue.add(currNode.left);
            }

            // For the right child
            i++;
            if(i >= ip.length)
                break;

            currVal = ip[i];

            // If the right child is not null
            if(!currVal.equals("N")) {

                // Create the right child for the current node
                currNode.right = new Node(Integer.parseInt(currVal));
```

```
                // Push it to the queue
                queue.add(currNode.right);
            }
            i++;
        }

        return root;
    }
    static void printInorder(Node root)
    {
        if(root == null)
            return;

        printInorder(root.left);
        System.out.print(root.data+" ");

        printInorder(root.right);
    }

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

            int t=Integer.parseInt(br.readLine());

            while(t-- > 0){
                String s = br.readLine();
                Node root = buildTree(s);
                Tree g = new Tree();
                ArrayList<Integer> diagonalNode = g.diagonal(root);
                for(int i = 0 ;i<diagonalNode.size();i++){
                    System.out.print(diagonalNode.get(i)+ " ");
                }
                System.out.println();
            }
    }
}

} Driver Code Ends

User function Template for Java
/* Node is defined as
class Node{
    int data;
    Node left;
    Node right;
    Node(int data){
        this.data = data;
        left=null;
        right=null;
    }
}*/
class Tree
{
    class node {
        Node  root;
        int level;
```

```
   node (Node root,  int level)
    {
        this.root=root;
        this.level=level;

    }
}
 public ArrayList<Integer> diagonal(Node root)
  {
       //add your code here.
      ArrayList <Integer> ans=new ArrayList <>();

      TreeMap< Integer  , ArrayList <Integer > > map = new TreeMap<>();

      Queue<node > q=new LinkedList ();

      q.add(new node (root ,0));

      while(!q.isEmpty())
      {

          node peek= q.peek();
          q.poll();
          Node peekNode=peek.root;

          if(map.containsKey (peek.level))
          {
              map.get(peek.level).add(peekNode.data);

          }
          else{
              ArrayList <Integer> list =new ArrayList <>();
              list.add(peekNode.data);

              map.put(peek.level,list);
          }
          if(peekNode.left!=null)
          {
              node val= new node (peekNode.left,peek.level+1);
              q.add(val);
          }

          if(peekNode.right!=null)
          {
              node val=new node (peekNode.right,peek.level);

              q.add(val);

          }

          System.out.println(map);


      }

      for(int key : map.keySet())
      {
```

```
                ArrayList <Integer> list  =map.get(key);
                for(int value :list  )
                {
                    ans.add(value);

                }
            }


            return ans ;

             Queue<Node> q=new ArrayDeque<>();
             ArrayList<Integer> list=new ArrayList<Integer>();
             q.add(root);
             while(!q.isEmpty())
             {
               Node temp=q.poll();
               while(temp!=null)
               {
                list.add(temp.data);
                if(temp.left!=null)
                {
                  q.add(temp.left);
                }
                 temp=temp.right;
               }
             }
             return list;

        }
}
```