Path Sum

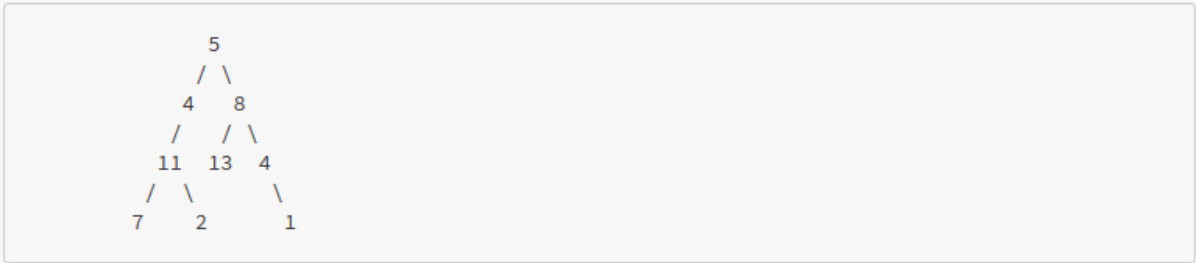
Asked in: Microsoft Yahoo Amazon

Suggest Edit | Bookmark

Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.

Example :

Given the below binary tree and `sum = 22`,



return `true`, as there exist a root-to-leaf path `5->4->11->2` which sum is `22`.

Return `0 / 1` (0 for false, 1 for true) for this problem

```
public class Solution {
    public int hasPathSum(TreeNode A, int B) {
        Stack<TreeNode> st=new Stack<TreeNode>();
        st.push(A);
        while(!st.isEmpty()){
            TreeNode cur=st.pop();
            if(cur.left==null && cur.right==null ){
                if(cur.val==B){
                    return 1;
                }
            }
            if(cur.left!=null){
                cur.left.val+=cur.val;
                st.push(cur.left);
            }

            if(cur.right!=null){
                cur.right.val+=cur.val;
                st.push(cur.right);
            }
        }
        return 0;
    }
}
```

Inorder Traversal

Asked in: Amazon Microsoft

Suggest Edit | Bookmark

Given a binary tree, return the inorder traversal of its nodes' values.

Example :

Given binary tree

```
   1
    \
     2
    /
   3
```

return `[1,3,2]`.

Using recursion is not allowed.

See Expected Output

```
public class Solution {
    public ArrayList<Integer> inorderTraversal(TreeNode A) {
     ArrayList<Integer> result=new ArrayList<Integer>();
     Stack<TreeNode> st=new Stack<TreeNode>();
     TreeNode cur=A;
     while(cur!=null || !st.isEmpty()){
        while(cur!=null){
            st.push(cur);
            cur=cur.left;
        }
        cur=st.pop();
        result.add(cur.val);
        cur=cur.right;
     }
     return result;
    }
}
```

Postorder Traversal

Asked in: Amazon Microsoft

Suggest Edit Bookmark

Given a binary tree, return the postorder traversal of its nodes' values.

Example :

Given binary tree

```
   1
    \
     2
    /
   3
```

return `[3,2,1]`.

Using recursion is not allowed.

```
public class Solution {
    public ArrayList<Integer> postorderTraversal(TreeNode A) {
        ArrayList<Integer> result=new ArrayList<Integer>();
        Stack<TreeNode> st=new Stack<TreeNode>();
        st.push(A);
        while(!st.isEmpty()){
            TreeNode temp=st.pop();
            result.add(0,temp.val);
            if(temp.left!=null){
                st.push(temp.left);
            }
            if(temp.right!=null){
                st.push(temp.right);
            }
        }
        return result;
    }
}
```

Preorder Traversal

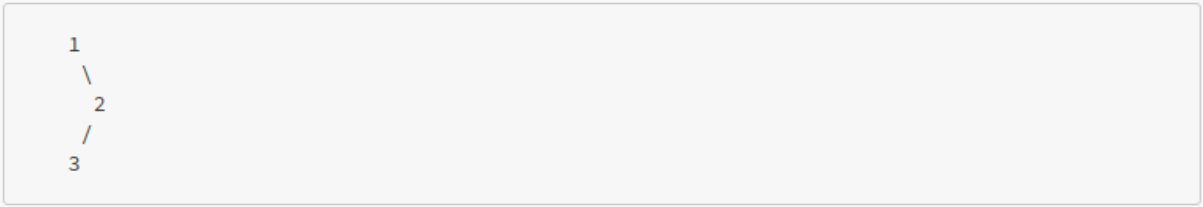
Asked in: Amazon Microsoft

Suggest Edit | Bookmark

Given a binary tree, return the preorder traversal of its nodes' values.

Example :

Given binary tree



return `[1,2,3]`.

Using recursion is not allowed.

```
public class Solution {
    public ArrayList<Integer> preorderTraversal(TreeNode A) {
        if(A==null){
            return null;
        }
        ArrayList<Integer> result=new ArrayList<Integer>();
        Stack<TreeNode> st=new Stack<TreeNode>();
        st.push(A);
        while(!st.isEmpty()){
            TreeNode temp=st.pop();
            result.add(temp.val);
            if(temp.right!=null){
                st.push(temp.right);
            }
            if(temp.left!=null){
                st.push(temp.left);
            }
        }
        return result;
    }
}
```

Vertical Order traversal of Binary Tree

Suggest Edit | Bookmark

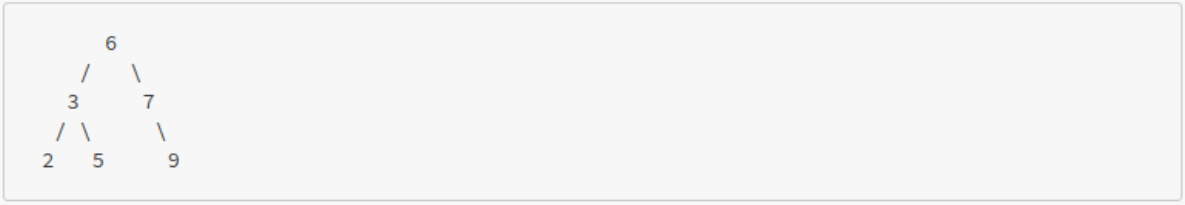
Asked in: Amazon

Problem Setter: yashpal1995 Problem Tester: RAMBO_tejasv

Given a binary tree, print a vertical order traversal of it.

Example :

Given binary tree:



returns

```
[
    [2],
    [3],
    [6 5],
    [7],
    [9]
]
```

```
public class Solution {
    public ArrayList<ArrayList<Integer>> verticalOrderTraversal(TreeNode A) {
        ArrayList<ArrayList<Integer>> res=new ArrayList<ArrayList<Integer>>();
        if(A==null){
            return res;
        }

        HashMap<Integer, ArrayList<Integer>> hm=new HashMap<Integer,ArrayList<Integer>>();
        LinkedList<TreeNode> q=new LinkedList<TreeNode>();
        LinkedList<Integer> level=new LinkedList<Integer>();
        q.offer(A);
        level.offer(0);
        int minLevel=0;
        int maxLevel=0;
        while(!q.isEmpty()){
            TreeNode p=q.poll();
            int l=level.poll();
            if(l<minLevel){
                minLevel=l;
            }
            else if(l>maxLevel){
                maxLevel=l;
            }
            if(hm.containsKey(l)){
                hm.get(l).add(p.val);
            }
```

```
            else{
            ArrayList<Integer> list = new ArrayList<Integer>();
            list.add(p.val);
            hm.put(l, list);
            }
            if(p.left!=null){
                q.offer(p.left);
                level.offer(l-1);
            }
            if(p.right!=null){
                q.offer(p.right);
                level.offer(l+1);
            }

        }
        for(int i=minLevel;i<=maxLevel;i++){
            if(hm.containsKey(i)){
                res.add(hm.get(i));
            }
        }

        return res;
    }
}
```

Identical Binary Trees

Asked in: Amazon

Suggest Edit | Bookmark

Given two binary trees, write a function to check if they are equal or not.

Two binary trees are considered equal if they are structurally identical and the nodes have the same value.

Return `0 / 1` (0 for false, 1 for true) for this problem

Example :

```
Input :

   1       1
  / \     / \
 2   3   2   3

Output :
  1 or True
```

```
public class Solution {
    public int isSameTree(TreeNode A, TreeNode B) {
        if(A==null && B==null){
            return 1;
        }

        else if(A==null || B==null){
            return 0;
        }
        Stack<TreeNode> a=new Stack<TreeNode>();
        Stack<TreeNode> b=new Stack<TreeNode>();
        a.push(A);
        b.push(B);

        while(!a.isEmpty()){
            try{
            TreeNode cur1=a.pop();
            TreeNode cur2=b.pop();
            int v1=cur1.val;
            int v2=cur2.val;
            if(v1!=v2){
                return 0;
            }


                if(cur1.left!=null){
                a.push(cur1.left);
                }
                if(cur1.right!=null){
```

```
                a.push(cur1.right);
            }
            if(cur2.left!=null){
            b.push(cur2.left);
            }
            if(cur2.right!=null){
                b.push(cur2.right);
            }

        }
            catch(Exception e){
              return 0;
            }
        }

    return 1;









  }
}
```

Symmetric Binary Tree

Asked in: Amazon

Suggest Edit | Bookmark

Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).

Example :

```
    1
   / \
  2   2
 / \ / \
3  4 4  3
```

The above binary tree is symmetric.
But the following is not:

```
    1
   / \
  2   2
   \   \
   3    3
```

Return `0 / 1` (0 for false, 1 for true) for this problem

```
public class Solution {
    public int isSymmetric(TreeNode A) {
      if(A==null){
         return 1;
      }
      Queue<TreeNode> q=new LinkedList<TreeNode>();
      try{
         q.add(A.left);
         q.add(A.right);
         while(!q.isEmpty()){
            TreeNode l=q.poll();
            TreeNode r=q.poll();
            if(l==null && r==null){
               continue;
            }
            q.add(l.left);
            q.add(r.right);
            q.add(l.right);
            q.add(r.left);
         }
       }
       catch(Exception e){
          return 0;
       }
```

```
        return 1;
    }
}
```

Invert the Binary Tree

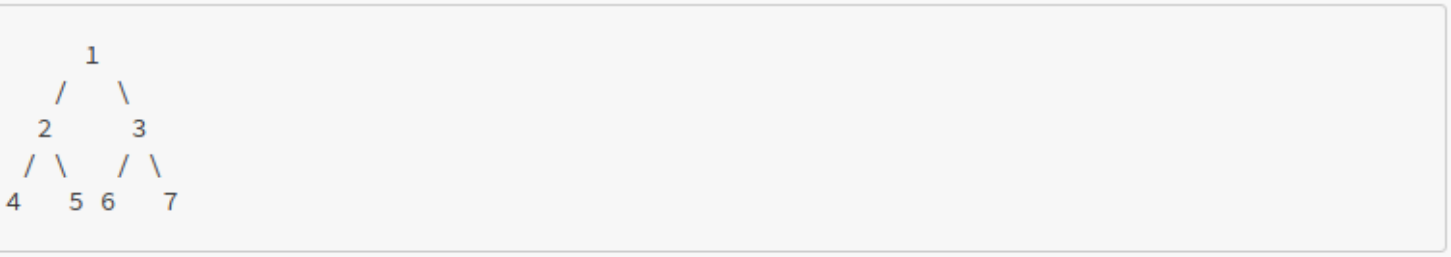
Asked in: Google

Suggest Edit | Bookmark

Given a binary tree, invert the binary tree and return it.
Look at the example for more details.

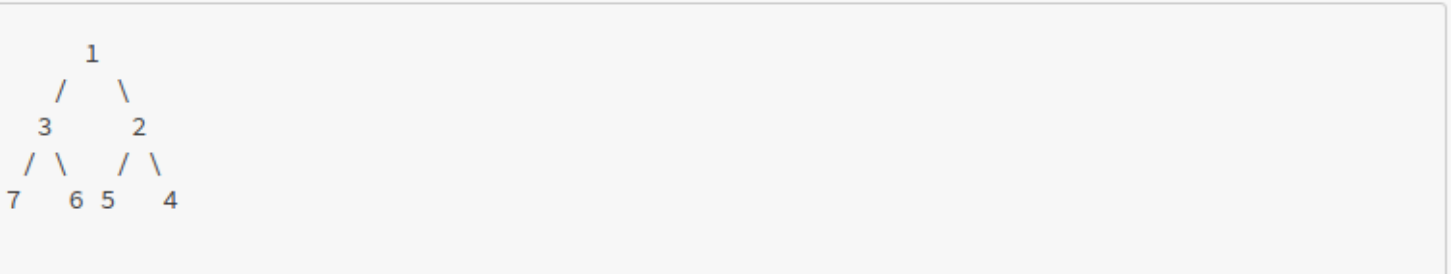
Example :

Given binary tree



```
     1
   /   \
  2     3
 / \   / \
4   5 6   7
```

invert and return



```
     1
   /   \
  3     2
 / \   / \
7   6 5   4
```

```
public class Solution {
    public TreeNode invertTree(TreeNode A) {
        if(A==null){
            return null;
        }
        Queue<TreeNode> q=new LinkedList<TreeNode>();
        q.add(A);
        while(!q.isEmpty()){
            TreeNode cur=q.poll();
            TreeNode temp=cur.left;
            cur.left=cur.right;
            cur.right=temp;
            if(cur.left!=null){
                q.add(cur.left);
            }
```

```
            if(cur.right!=null){
                q.add(cur.right);
            }
        }
        return A;
    }
}
```

Order of People Heights

Asked in: Google

Suggest Edit | Bookmark

You are given the following :

1. A positive number `N`
2. `Heights` : A list of `heights of N persons` standing in a `queue`
3. `Infronts` : A list of numbers corresponding to each person `(P)` that gives the **number of persons** who are **taller** than `P` and standing in front of `P`

You need to `return` list of actual order of persons's height

Consider that heights will be unique

Example

```
Input :
Heights: 5 3 2 6 1 4
InFronts: 0 1 2 0 3 2
```

```
Output :
actual order is: 5 3 2 1 6 4
```

So, you can see that for the person with height 5, there is no one taller than him who is in front of him, and hence `Infronts` has 0 for him.

For person with height 3, there is 1 person (Height : 5) in front of him who is taller than him.

You can do similar inference for other people in the list.

```
public class Solution {
    public ArrayList<Integer> order(ArrayList<Integer> A, ArrayList<Integer> B) {
      ArrayList<Integer> res=new ArrayList<Integer>();
      TreeMap<Integer, Integer> tm=new TreeMap<Integer,Integer>();
      if(A==null || B==null || A.size()!=B.size()){
         return res;
      }
      for(int i=0;i<A.size();i++){
         tm.put(A.get(i),B.get(i));
      }
      boolean flag=true;
      while(!tm.isEmpty()){
         int height=tm.lastKey();
         int order=tm.get(height);
         tm.remove(height);
         if(res.size()==0){
            res.add(0,height);
         }
```

```
            else if(order==0){
                res.add(0,height);
            }
            else{
                res.add(order,height);
            }

        }
        return res;

    }
}
```

Flatten Binary Tree to Linked List

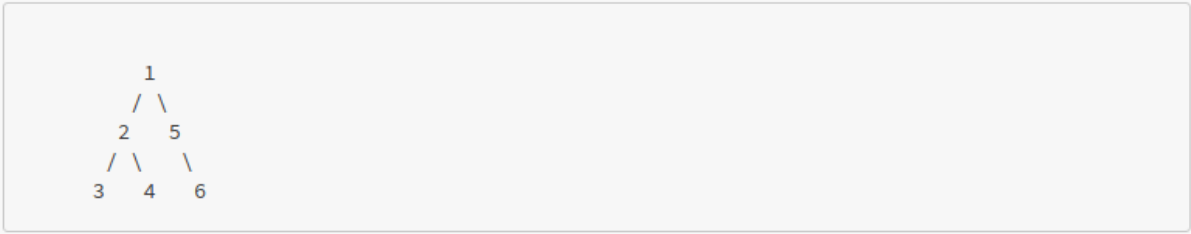
Asked in: Adobe Amazon Microsoft

Suggest Edit | Bookmark

Given a binary tree, flatten it to a linked list in-place.

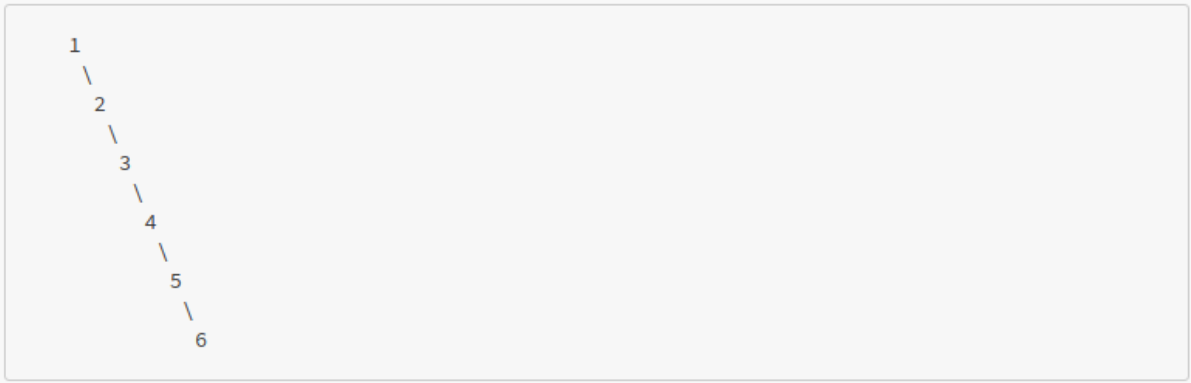
Example :

Given



The flattened tree should look like:

The flattened tree should look like:



Note that the left child of all nodes should be `NULL`.

See Expected Output

```
public class Solution {
    public TreeNode flatten(TreeNode a) {
        if(a==null){
            return null;
        }
        TreeNode result=new TreeNode(0);
        TreeNode prev=result;
        result.right=prev;
        Stack<TreeNode> st=new Stack<TreeNode>();
        st.push(a);
        while(!st.isEmpty()){
            TreeNode temp=st.pop();
            prev.right=temp;
```

```
            prev.left=null;
            prev=prev.right;
            if(temp.right!=null){
                st.push(temp.right);
            }
            if(temp.left!=null){
                st.push(temp.left);
            }
        }
        return result.right;
    }
}
```

Least Common Ancestor

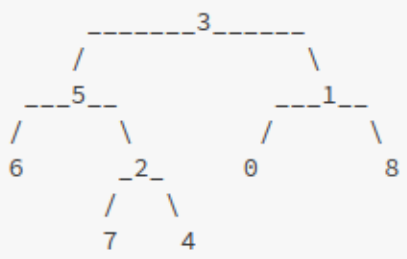
Asked in: Facebook Adobe Microsoft Amazon Google

Suggest Edit Bookmark

Find the lowest common ancestor in an unordered binary tree given two values in the tree.

> *“* ***Lowest common ancestor :*** *the lowest common ancestor (LCA) of two nodes* `v` *and* `w` *in a tree or directed acyclic graph (DAG) is the lowest (i.e. deepest) node that has both* `v` *and* `w` *as descendants.* *”*

Example :



For the above tree, the LCA of nodes `5` and `1` is `3`.

```
public class Solution {
    public int lca(TreeNode node, int n1, int n2) {
        TreeNode n=helper(node,n1,n2);
        return n!=null?n.val-1;


    }
    public TreeNode helper(TreeNode root, int n1, int n2){
        if(root == null){
            return null;
        }
        if(root.val == n1 || root.val == n2){
            return root;
        }

        TreeNode left = helper(root.left, n1, n2);
        TreeNode right = helper(root.right, n1, n2);

        if(left != null && right != null){
            return root;
        }
```

```
        return left != null ? left : right;


    }
}
```

Sum Root to Leaf Numbers

Asked in: Google Microsoft

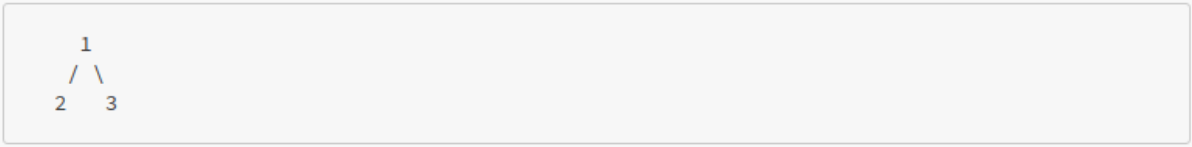
Suggest Edit | Bookmark

Given a binary tree containing digits from 0-9 only, each root-to-leaf path could represent a number.

An example is the root-to-leaf path `1->2->3` which represents the number `123`.

Find the total sum of all root-to-leaf numbers % 1003.

Example :



The root-to-leaf path `1->2` represents the number `12`.
The root-to-leaf path `1->3` represents the number `13`.

Return the sum = `(12 + 13) % 1003 = 25 % 1003 = 25`.

See Expected Output

```
public class Solution {
    public int sumNumbers(TreeNode A) {
        if(A==null){
            return 0;
        }
        return findSum(A,0)%1003;
    }
    public int findSum(TreeNode A,int s){
        if(A==null){
            return 0;
        }
        if(A.left==null && A.right==null){
            return (A.val+s*10)%1003;
        }
        s=(A.val+s*10)%1003;
        return findSum(A.left,s)%1003+findSum(A.right,s)%1003;
```

```
    }

}
```

Min Depth of Binary Tree

Asked in: Facebook Amazon

Suggest Edit | Bookmark

Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

> ***NOTE :*** *The path has to end on a leaf node.*

Example :

```
    1
   /
  2
```

min depth = 2.

```
public class Solution {
    public int minDepth(TreeNode A) {
        if(A==null){
            return 0;
        }
        int l=minDepth(A.left);
        int r=minDepth(A.right);
        return (l == 0 || r == 0) ? l+r+1:Math.min(l,r);
    }
}
```

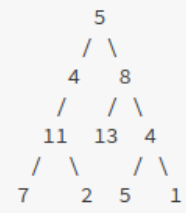
Root to Leaf Paths With Sum

Asked in: Microsoft Yahoo Amazon

Suggest Edit | Bookmark

Given a binary tree and a sum, find all root-to-leaf paths where each path's sum equals the given sum.

For example:
Given the below binary tree and sum = 22,



return

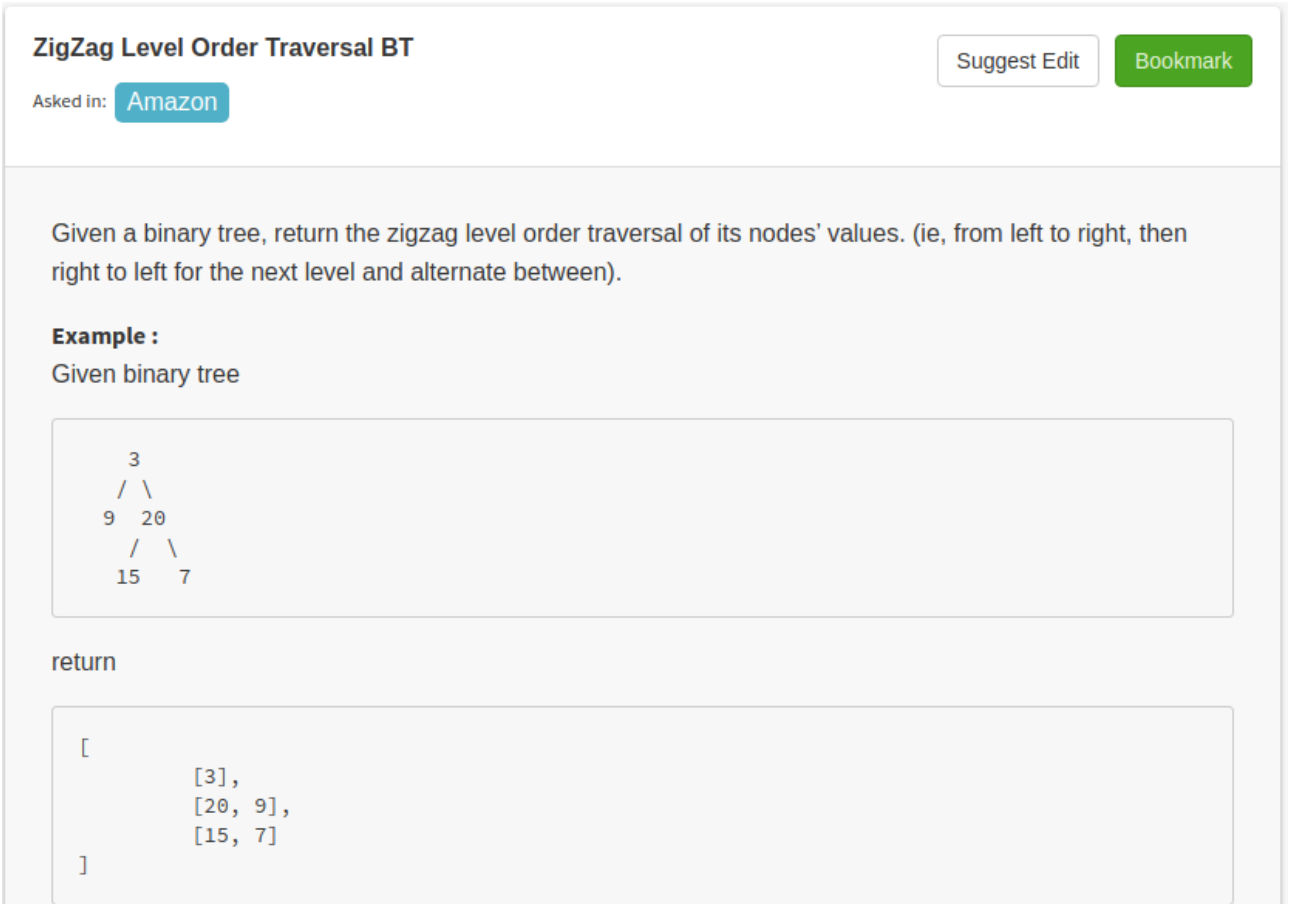
```
[
   [5,4,11,2],
   [5,8,4,5]
]
```

```
public class Solution {
   ArrayList<ArrayList<Integer>> result=new ArrayList<ArrayList<Integer>>();
   public ArrayList<ArrayList<Integer>> pathSum(TreeNode A, int B) {

      if(A==null){
         return result;
      }
      ArrayList<Integer> temp=new ArrayList<Integer>();
      helper(A,B,temp);
      return result;
   }
   public void helper(TreeNode A,int B,ArrayList<Integer>temp){
      if(A==null){
         return;
      }

      temp.add(A.val);
      int remaining=A.val;
      if(A.left==null && A.right==null){
         if(B-remaining==0){
            result.add(new ArrayList<Integer>(temp));
         }
      }
      helper(A.left,B-remaining,temp);
      helper(A.right,B-remaining,temp);
      temp.remove(temp.size()-1);
```

```
    }
}
```



ZigZag Level Order Traversal BT

Asked in: Amazon

Suggest Edit | Bookmark

Given a binary tree, return the zigzag level order traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).

Example :

Given binary tree

```
    3
   / \
  9  20
    /  \
   15   7
```

return

```
[
         [3],
         [20, 9],
         [15, 7]
]
```

```
public class Solution {
    public ArrayList<ArrayList<Integer>> zigzagLevelOrder(TreeNode A) {
        ArrayList<ArrayList<Integer>> res=new ArrayList<ArrayList<Integer>>();
        if(A==null){
            return res;
        }
        Stack<TreeNode> st1=new Stack<TreeNode>();
        Stack<TreeNode> st2=new Stack<TreeNode>();
        st1.push(A);
        while(!st1.isEmpty() || !st2.isEmpty()){
            ArrayList<Integer> temp1=new ArrayList<Integer>();
            ArrayList<Integer> temp2=new ArrayList<Integer>();
            while(!st1.isEmpty()){
                TreeNode temp=st1.pop();
                temp1.add(temp.val);
                if(temp.left!=null){
                    st2.push(temp.left);
                }
                if(temp.right!=null){
                    st2.push(temp.right);
                }
            }
```

```
            if(temp1.size()>0){
                res.add(temp1);
            }

            while(!st2.isEmpty()){
                TreeNode temp=st2.pop();
                temp2.add(temp.val);
                if(temp.right!=null){
                    st1.push(temp.right);
                }
                if(temp.left!=null){
                    st1.push(temp.left);
                }
            }
            if(temp2.size()>0){
                res.add(temp2);
            }

        }
        return res;


    }
}
```