Kartikey

TREES

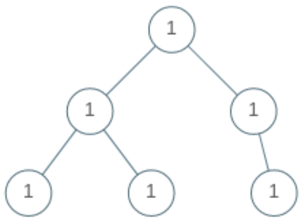
965. Univalued Binary Tree

Easy 1476 57 Add to List Share

A binary tree is **uni-valued** if every node in the tree has the same value.

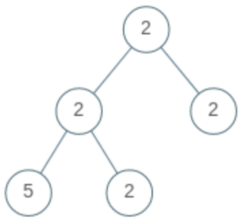
Given the `root` of a binary tree, return `true` *if the given tree is **uni-valued**, or* `false` *otherwise.*

Example 1:



```
Input: root = [1,1,1,1,1,null,1]
Output: true
```

Example 2:



```
Input: root = [2,2,2,5,2]
Output: false
```

```
class Solution {
public:
    bool get(TreeNode* root,int val)
    {
        if(root==NULL)
            return true;
```

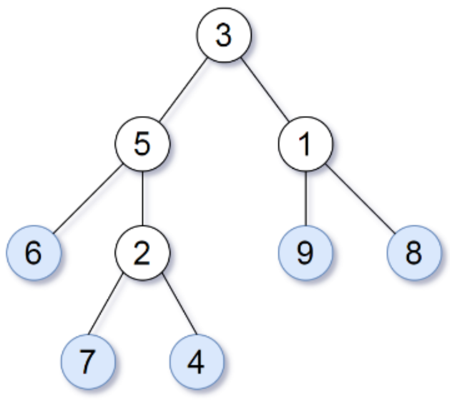
```
         bool lft= get(root->left,val);
          bool rght=get(root->right,val);

           if(root->val==val && lft &&rght)
              return true;
      return false;

      }
      bool isUnivalTree(TreeNode* root) {
          if(root==NULL)
              return true;

          return get(root,root->val);
      }
  };
```

https://leetcode.com/problems/univalued-binary-tree/

872. Leaf-Similar Trees

Easy 1921 57 Add to List Share

Consider all the leaves of a binary tree, from left to right order, the values of those leaves form a **leaf value sequence**.



For example, in the given tree above, the leaf value sequence is `(6, 7, 4, 9, 8)`.

Two binary trees are considered *leaf-similar* if their leaf value sequence is the same.

Return `true` if and only if the two given trees with head nodes `root1` and `root2` are leaf-similar.

```
class Solution {
public:
    void get(TreeNode* root,vector<int> &v)
    {
        if(root==NULL)
            return;

        if(root->left==NULL && root->right==NULL)
            v.push_back(root->val);

        get(root->left,v);
        get(root->right,v);

    }
    bool leafSimilar(TreeNode* root1, TreeNode* root2) {
        vector<int> tree1;
        vector<int> tree2;
        get(root1,tree1);
        get(root2,tree2);
        return tree1==tree2;
    }
};
```

https://leetcode.com/problems/leaf-similar-trees/

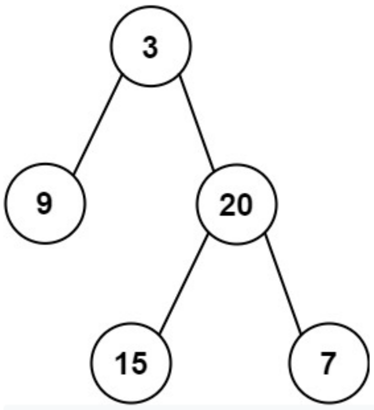
404. Sum of Left Leaves

Easy 3901 262 Add to List Share

Given the `root` of a binary tree, return *the sum of all left leaves.*

A **leaf** is a node with no children. A **left leaf** is a leaf that is the left child of another node.

Example 1:



```
Input: root = [3,9,20,null,null,15,7]
Output: 24
Explanation: There are two left leaves in the binary tree, with values 9 and 15 respectively.
```

Example 2:

```
Input: root = [1]
Output: 0
```

```
class Solution {
public:
    void solve(TreeNode* root,int &sum)
    {
        if(root==NULL)
            return;

        if(root->left)
        {
            if(root->left->left==NULL && root->left->right==NULL)
            {
                sum+=root->left->val;

            }
        }
        solve(root->left,sum);
        solve(root->right,sum);
    }
    int sumOfLeftLeaves(TreeNode* root) {
        int sum=0;
        solve(root,sum);
        return sum;
    }
};
```

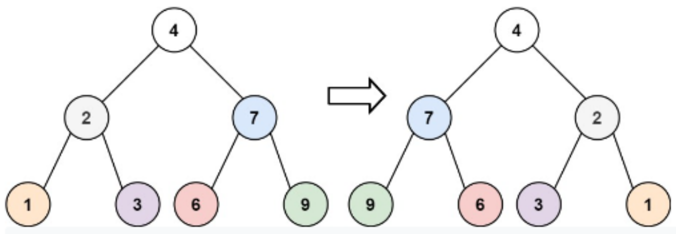
https://leetcode.com/problems/sum-of-left-leaves/

226. Invert Binary Tree

Easy 9983 137 Add to List Share

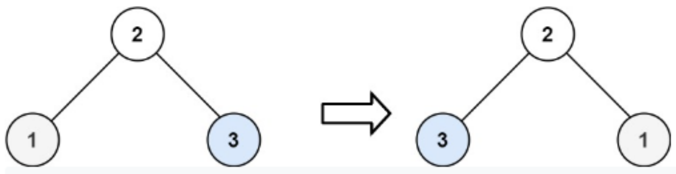
Given the `root` of a binary tree, invert the tree, and return *its root*.

Example 1:



```
Input: root = [4,2,7,1,3,6,9]
Output: [4,7,2,9,6,3,1]
```

Example 2:



```
Input: root = [2,1,3]
Output: [2,3,1]
```

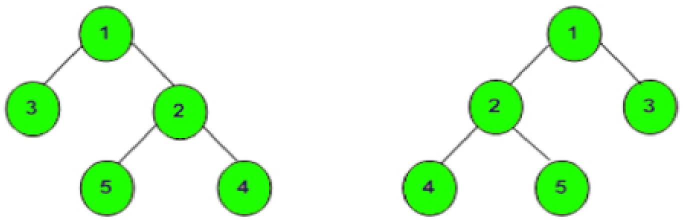
```
class Solution {
public:
    void solve(TreeNode* root)
    {
        if(root==NULL)
            return;

        solve(root->left);
        solve(root->right);

        TreeNode* temp=root->right;
        root->right=root->left;
        root->left=temp;
    }
    TreeNode* invertTree(TreeNode* root) {
        if(root==NULL)
            return NULL;
        TreeNode *t=root;
        solve(root);
        return t;


    }
};
```

https://leetcode.com/problems/invert-binary-tree/

Mirror Tree

Easy Accuracy: **63.43%** Submissions: **100k+** Points: **2**

Given a Binary Tree, convert it into its mirror.



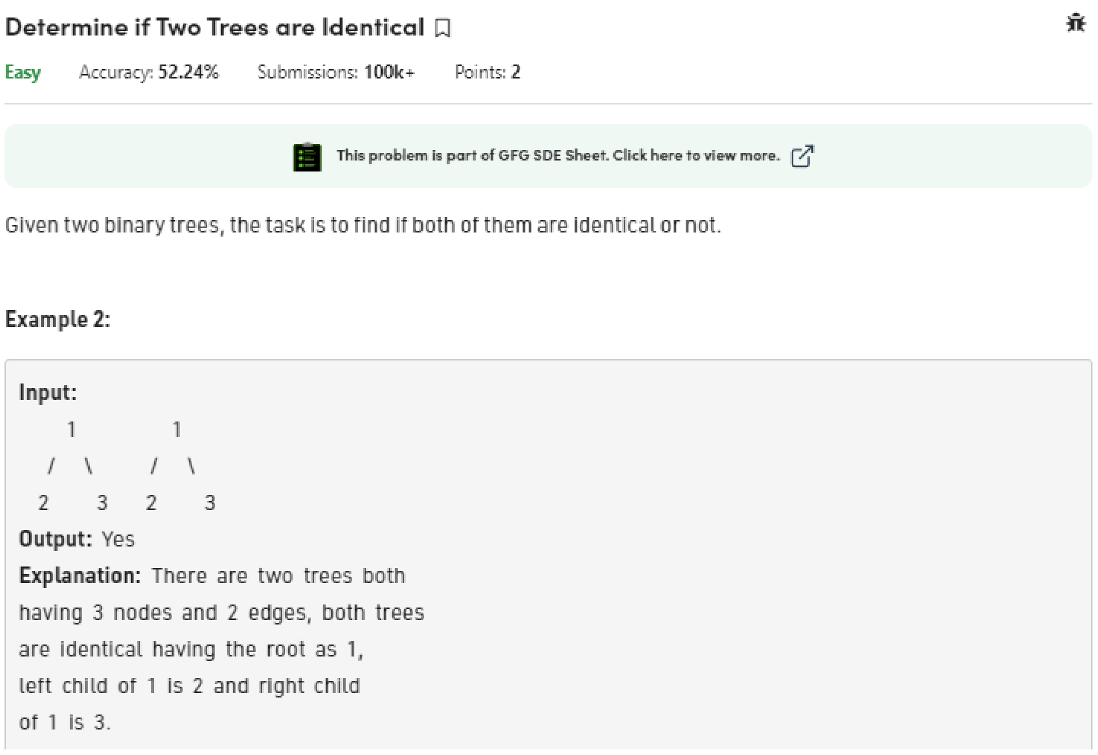
Mirror Trees

Example 1:

```
Input:
      1
    /  \
   2    3
Output: 3 1 2
Explanation: The tree is
   1    (mirror)  1
 /  \    =>      /  \
2    3          3    2
The inorder of mirror is 3 1 2
```

```
class Solution {
  public:
    // Function to convert a binary tree into its mirror tree.
    void mirror(Node* root) {
        // code here
        if(root==NULL)
        return ;
        mirror(root->left);
        mirror(root->right);

        Node *temp=root->left;
        root->left=root->right;
        root->right=temp;


        }
};
```

https://practice.geeksforgeeks.org/problems/mirror-tree/1



Determine if Two Trees are Identical

Easy   Accuracy: **52.24%**   Submissions: **100k+**   Points: **2**

This problem is part of GFG SDE Sheet. Click here to view more.

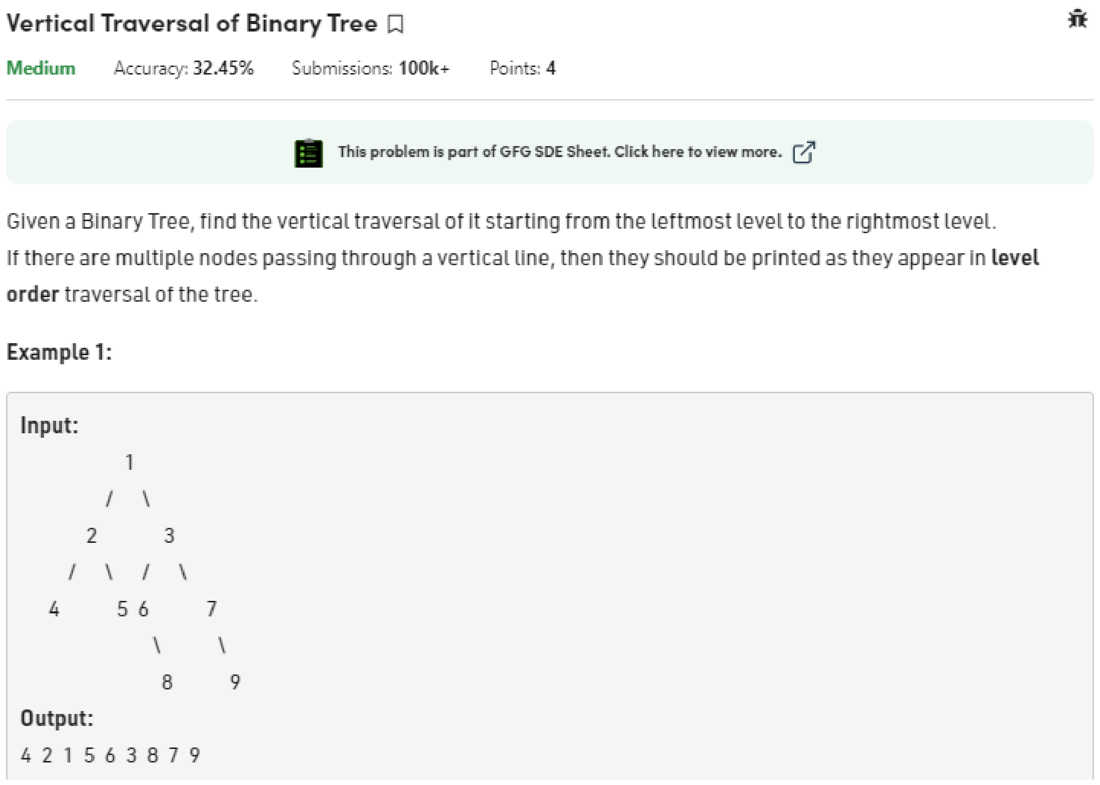
Given two binary trees, the task is to find if both of them are identical or not.

Example 2:

Input:
```
    1          1
  /   \      /   \
 2     3    2     3
```
Output: Yes
Explanation: There are two trees both having 3 nodes and 2 edges, both trees are identical having the root as 1, left child of 1 is 2 and right child of 1 is 3.

```
class Solution
{
    public:
    //Function to check if two trees are identical.

    bool isIdentical(Node *r1, Node *r2)
    {
        //Your Code here
        if(r1==NULL&&r2==NULL)
        return true;
         if(r1==NULL&&r2!=NULL)
        return false;
         if(r1!=NULL&&r2==NULL)
        return false;

        bool t=isIdentical(r1->left,r2->left);
        bool tr=isIdentical(r1->right,r2->right);
        if(r1->data==r2->data&&t&&tr)
        {
            return true;
        }
        return false;
    }
};
```

https://practice.geeksforgeeks.org/problems/determine-if-two-trees-are-identical/1


Vertical Traversal of Binary Tree

Medium Accuracy: 32.45% Submissions: 100k+ Points: 4

This problem is part of GFG SDE Sheet. Click here to view more.

Given a Binary Tree, find the vertical traversal of it starting from the leftmost level to the rightmost level.
If there are multiple nodes passing through a vertical line, then they should be printed as they appear in **level order** traversal of the tree.

Example 1:

Input:

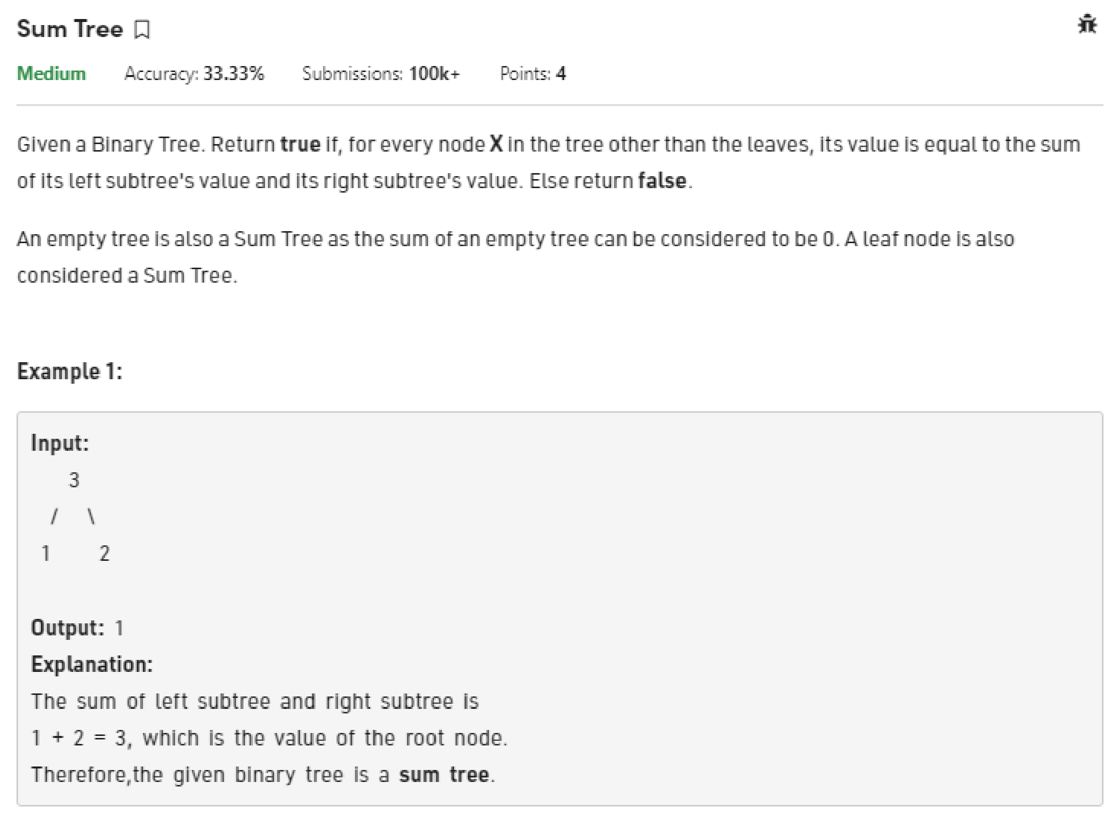
```
          1
        /   \
      2       3
    /   \   /   \
  4      5 6      7
            \      \
             8      9
```

Output:

```
4 2 1 5 6 3 8 7 9
```

```
public:
    //Function to find the vertical order traversal of Binary Tree.
    vector<int> verticalOrder(Node *root)
    {
        //Your code here
        //map hd,level,node
        map<int,map<int,vector<int>>> nodes;
        //queue node,hd,level
        queue<pair<Node*,pair<int,int>>> q;
        vector<int> ans;
        if(root==NULL)
        return ans;

        q.push(make_pair(root,make_pair(0,0)));

        while(!q.empty())
        {
            pair<Node *,pair<int,int>> temp=q.front();
            q.pop();
            Node* frontNode=temp.first;
            int hd=temp.second.first;
            int level=temp.second.second;

            nodes[hd][level].push_back(frontNode->data);


            if(frontNode->left)
            q.push(make_pair(frontNode->left,make_pair(hd-1,level+1)));
            if(frontNode->right)
            q.push(make_pair(frontNode->right,make_pair(hd+1,level+1)));


        }
        for(auto i:nodes)
        {
            for(auto j:i.second)
            {
                for(auto k:j.second)
                {
                    ans.push_back(k);
                }
            }
        }
        return ans;
    }
```

https://practice.geeksforgeeks.org/problems/print-a-binary-tree-in-vertical-order/1



Sum Tree

Medium   Accuracy: 33.33%   Submissions: 100k+   Points: 4

Given a Binary Tree. Return **true** if, for every node **X** in the tree other than the leaves, its value is equal to the sum of its left subtree's value and its right subtree's value. Else return **false**.

An empty tree is also a Sum Tree as the sum of an empty tree can be considered to be 0. A leaf node is also considered a Sum Tree.

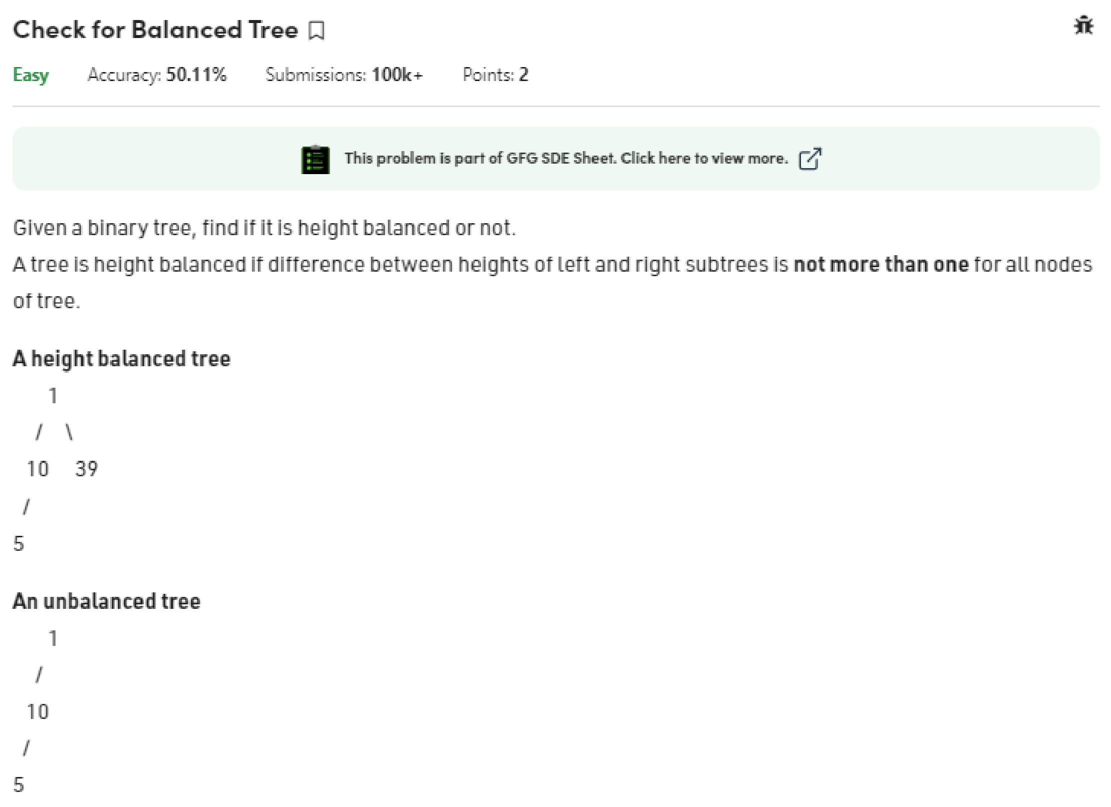
Example 1:

```
Input:
    3
  /   \
 1     2

Output: 1
Explanation:
The sum of left subtree and right subtree is
1 + 2 = 3, which is the value of the root node.
Therefore,the given binary tree is a sum tree.
```

```
// Should return true if tree is Sum Tree, else false
class Solution
{
    public:
    pair<bool,int> su(Node *root)
    {
        if(root==NULL)
        {
             pair<bool,int> p=make_pair(true,0);
             return p;
        }


        if(root->left==NULL && root->right==NULL)
        {
            pair<bool,int> p=make_pair(true,root->data);
            return p;
        }
        pair<bool,int> leftans=su(root->left);
        pair<bool,int> rightans=su(root->right);

        pair<bool,int> ans;
        int s=leftans.second+rightans.second;
        if(s==root->data && leftans.first && rightans.first)
        {ans.first=true;
        ans.second=2*root->data;
    }
    else
    ans.first=false;
```

```
        return ans;
    }
    bool isSumTree(Node* root)
    {
         // Your code here
         pair<bool,int> p=su(root);

         return p.first;


    }
};
```

https://practice.geeksforgeeks.org/problems/sum-tree/1?page=2&category[]=Tree&curated[]=7&sortBy=submissions



Check for Balanced Tree

Easy Accuracy: **50.11%** Submissions: **100k+** Points: **2**

This problem is part of GFG SDE Sheet. Click here to view more.

Given a binary tree, find if it is height balanced or not.
A tree is height balanced if difference between heights of left and right subtrees is **not more than one** for all nodes of tree.

A height balanced tree

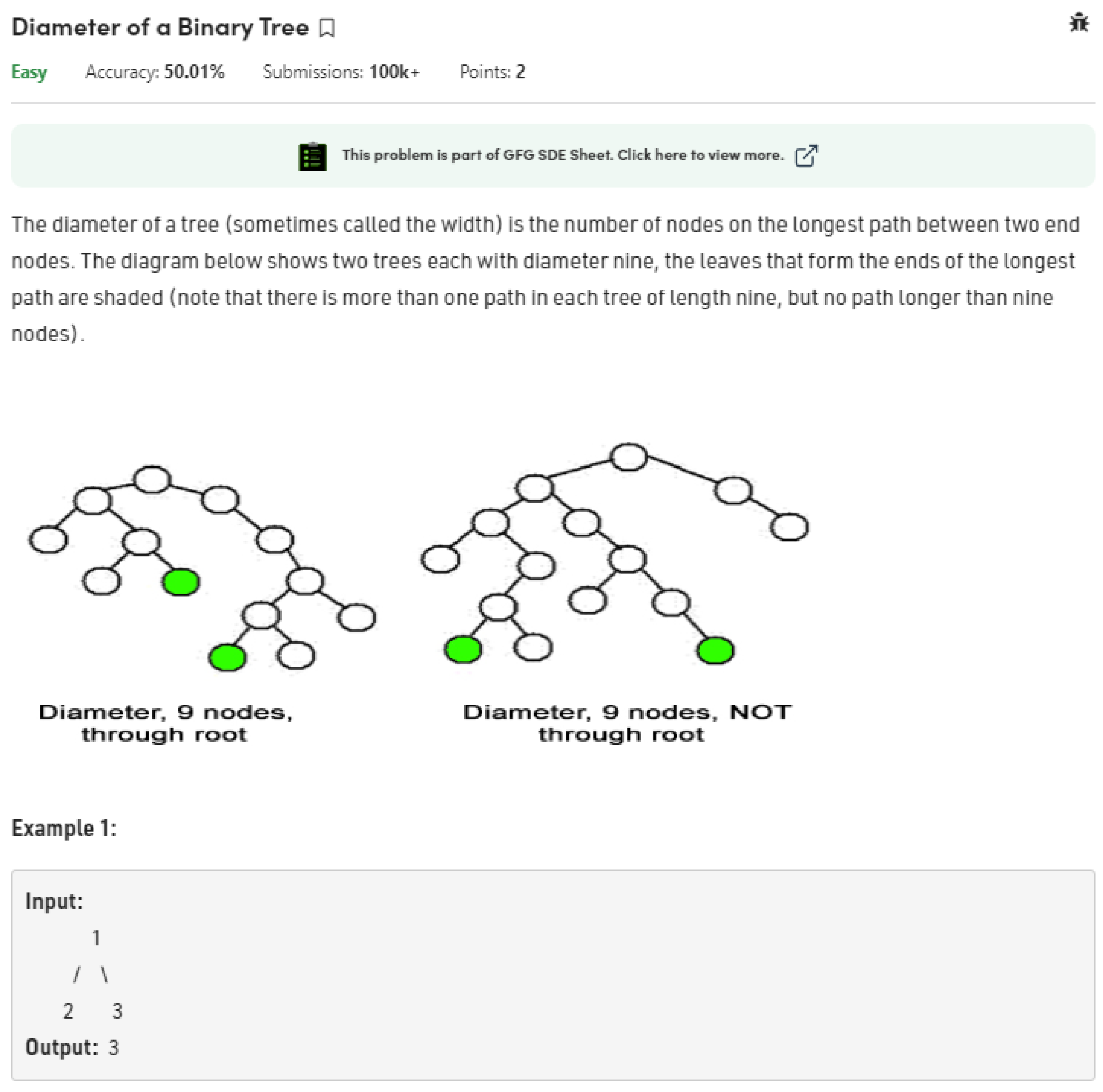
```
   1
  /   \
 10   39
 /
5
```

An unbalanced tree

```
   1
  /
 10
 /
5
```

```
class Solution{
    public:
    int height(Node *root)
    {
        if(root==NULL)
        return 0;

        return 1+max(height(root->left),height(root->right));
    }
    //Function to check whether a binary tree is balanced or not.
    bool isBalanced(Node *root)
    {
        //  Your Code here


        if(root==NULL)
        return true;
        int lefth=height(root->left);
        int righ=height(root->right);
        bool leftt=isBalanced(root->left);
        bool rightt=isBalanced(root->right);
        if(abs(righ-lefth)<=1&&rightt&&leftt)
        return true;

        return false;
    }
};
```

https://practice.geeksforgeeks.org/problems/check-for-balanced-tree/1?page=1&category[]=Tree&curated[]=7&sortBy=submissions



Diameter of a Binary Tree

Easy Accuracy: 50.01% Submissions: 100k+ Points: 2

This problem is part of GFG SDE Sheet. Click here to view more.

The diameter of a tree (sometimes called the width) is the number of nodes on the longest path between two end nodes. The diagram below shows two trees each with diameter nine, the leaves that form the ends of the longest path are shaded (note that there is more than one path in each tree of length nine, but no path longer than nine nodes).

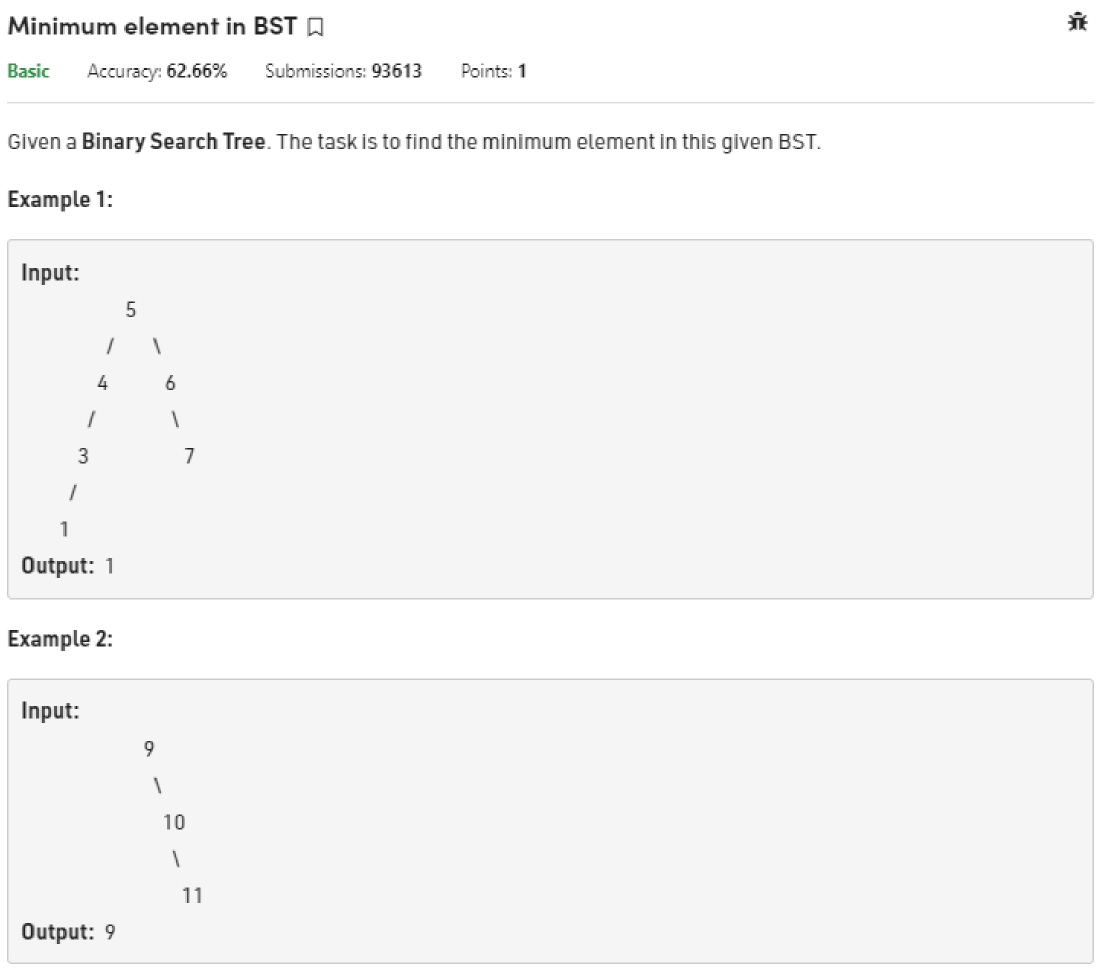
Diameter, 9 nodes, through root
Diameter, 9 nodes, NOT through root

Example 1:

```
Input:
      1
    /  \
   2    3
Output: 3
```

```
class Solution {
  public:
  int height(Node *root)
  {
      if(root==NULL)
      {
          return 0;
      }

      return 1+max(height(root->left),height(root->right));
  }
    // Function to return the diameter of a Binary Tree.
    int diameter(Node* root) {
        // Your code here
        if(root==NULL)
        return 1;

        int ld=diameter(root->left);
        int rd=diameter(root->right);
        int lefth=height(root->left);
        int righth=height(root->right);

        int h=lefth+righth+1;
        return max(h,max(ld,rd));
    }
};
```

https://practice.geeksforgeeks.org/problems/diameter-of-binary-tree/1?page=1&category[]=Tree&curated[]=7&sortBy=submissions



Minimum element in BST

Basic   Accuracy: 62.66%   Submissions: 93613   Points: 1

Given a **Binary Search Tree**. The task is to find the minimum element in this given BST.

Example 1:

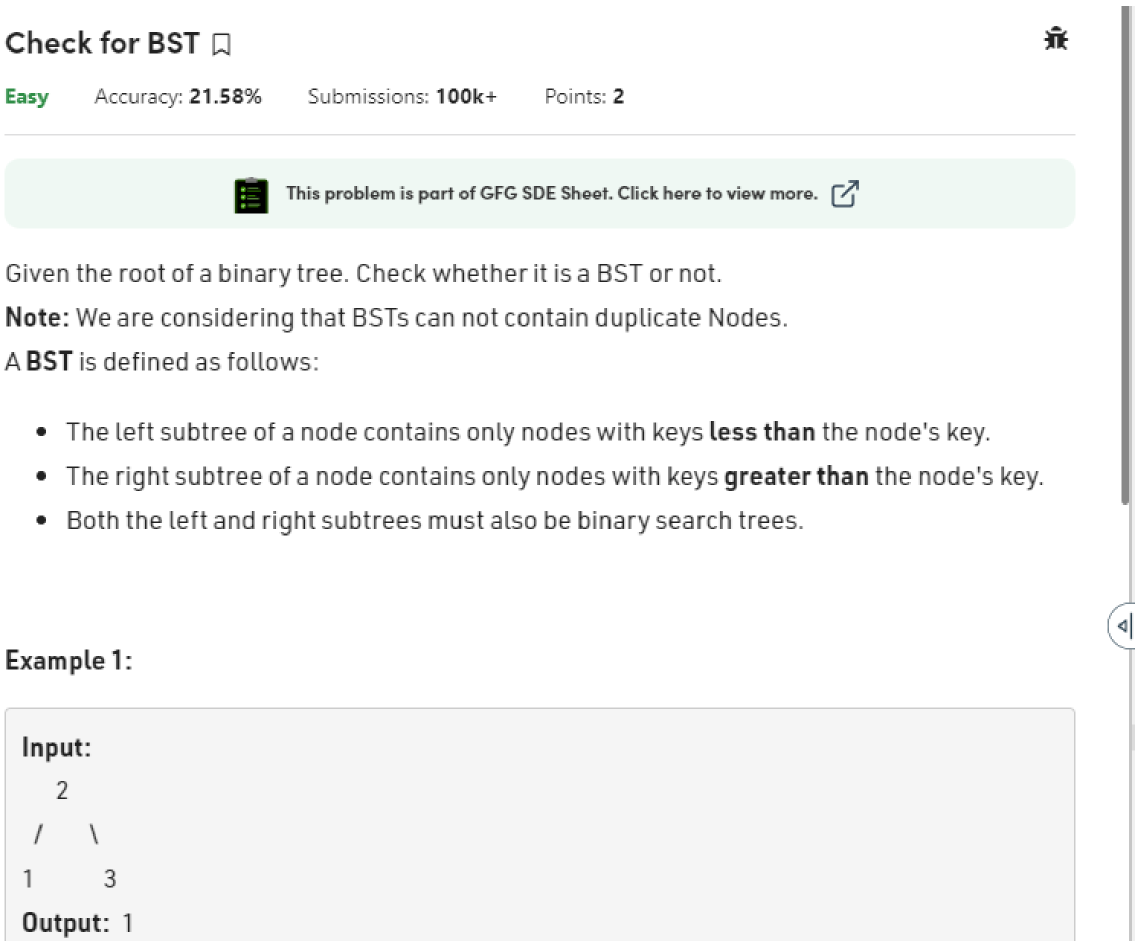
```
Input:
           5
         /   \
        4     6
       /       \
      3         7
     /
    1
Output: 1
```

Example 2:

```
Input:
             9
              \
               10
                \
                 11
Output: 9
```

```
int minValue(Node* root) {
    // Code here

    if(root==NULL)
    return -1;
    if(root->left==NULL)
    return root->data;

    minValue(root->left);

}
```

https://practice.geeksforgeeks.org/problems/minimum-element-in-bst/1?page=1&category[]=Tree&curated[]=7&sortBy=submissions


Check for BST

Easy   Accuracy: **21.58%**   Submissions: **100k+**   Points: **2**

This problem is part of GFG SDE Sheet. Click here to view more.

Given the root of a binary tree. Check whether it is a BST or not.

Note: We are considering that BSTs can not contain duplicate Nodes.

A **BST** is defined as follows:

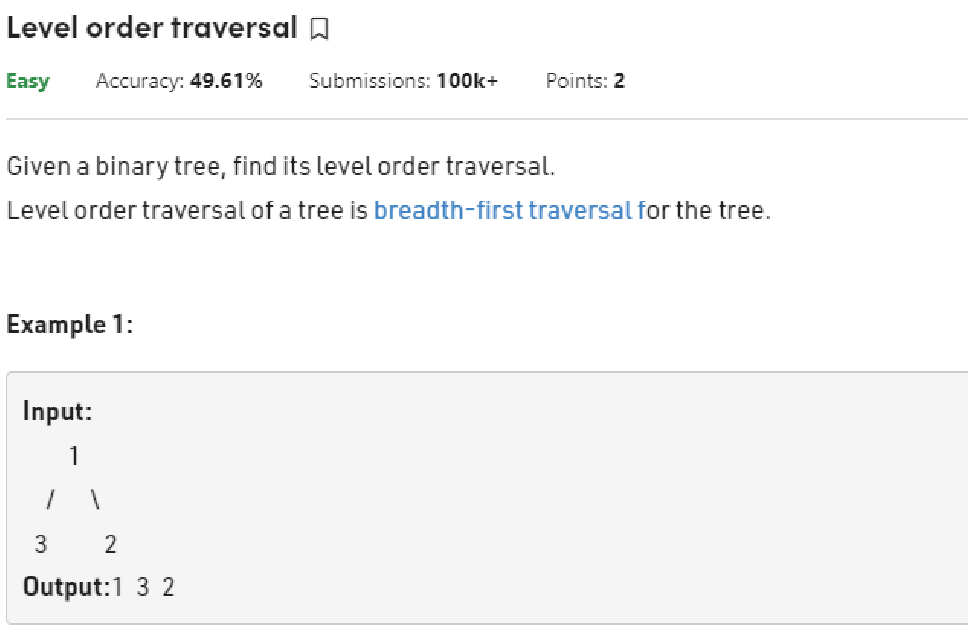
- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:

```
Input:
   2
 /    \
1       3
Output: 1
```

```
public:
    //Function to check whether a Binary Tree is BST or not.
    bool isBSTUtil( Node* root, Node*& prev)
{
    // traverse the tree in inorder fashion and
    // keep track of prev node
    if (root) {
        if (!isBSTUtil(root->left, prev))
            return false;

        // Allows only distinct valued nodes
        if (prev != NULL && root->data <= prev->data)
            return false;

        prev = root;

        return isBSTUtil(root->right, prev);
    }

    return true;
}
    bool isBST(Node* root)
    {
        // Your code here
        Node *prev=NULL;
       return  isBSTUtil(root,prev);



    }
```

https://practice.geeksforgeeks.org/problems/check-for-bst/1?page=1&category[]=Tree&curated[]=7&sortBy=submissions



Level order traversal

Easy Accuracy: **49.61%** Submissions: **100k+** Points: **2**

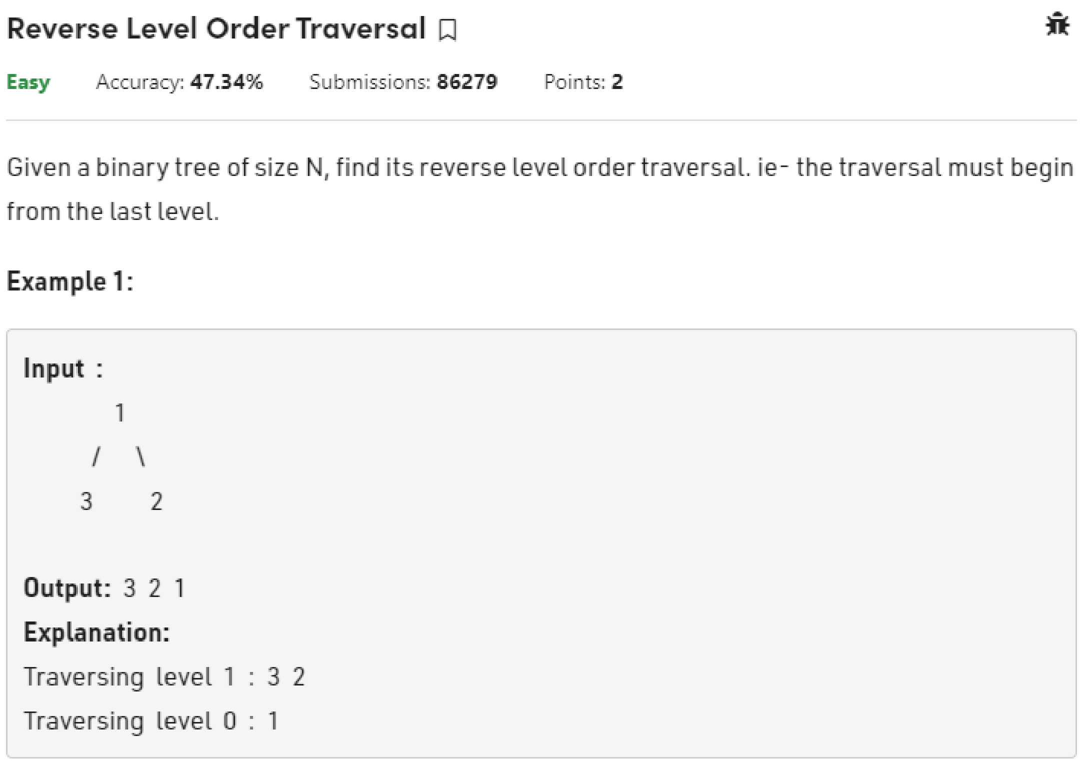
Given a binary tree, find its level order traversal.
Level order traversal of a tree is breadth-first traversal for the tree.

Example 1:

```
Input:
    1
  /   \
 3     2
Output:1 3 2
```

```
class Solution
{
    public:
    //Function to return the level order traversal of a tree.
    vector<int> levelOrder(Node* root)
    {
        vector<int> v;
      //Your code here
      if(root==NULL)
      return v;

      queue<Node*> q;
      q.push(root);
      while(!q.empty())
      {
          Node *temp=q.front();
          q.pop();
          v.push_back(temp->data);
          if(temp->left)
          q.push(temp->left);
          if(temp->right)
          q.push(temp->right);
      }
      return v;
    }
};
```

https://practice.geeksforgeeks.org/problems/level-order-traversal/1?page=1&category[]=Tree&curated[]=7&sortBy=submissions



Reverse Level Order Traversal

Easy Accuracy: **47.34%** Submissions: **86279** Points: **2**

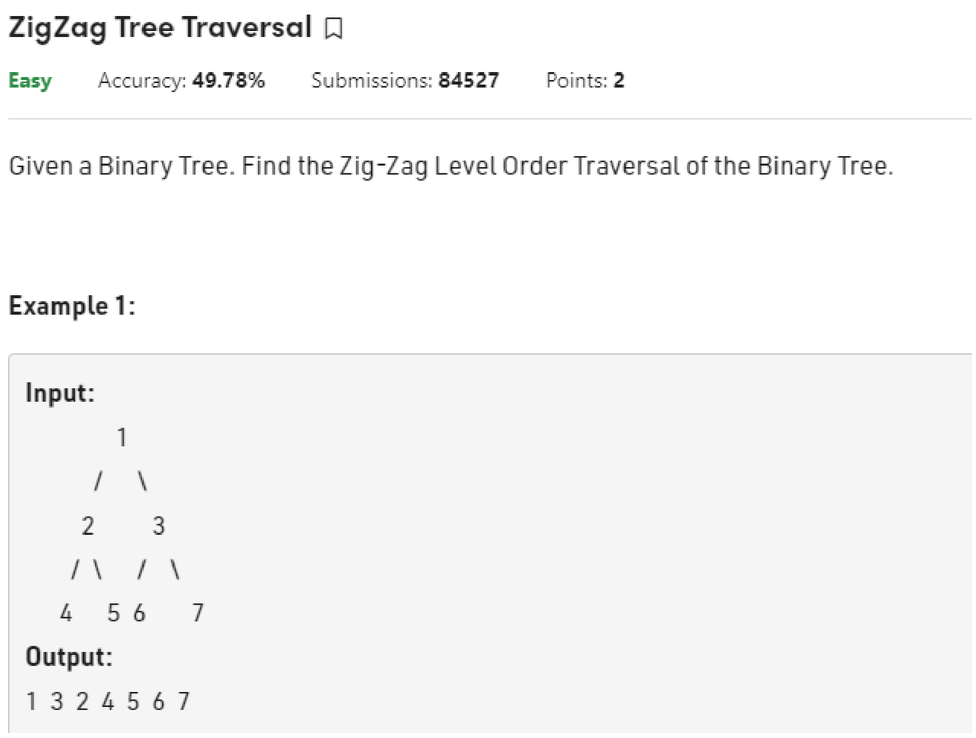
Given a binary tree of size N, find its reverse level order traversal. ie- the traversal must begin from the last level.

Example 1:

```
Input :
        1
      /   \
     3     2

Output: 3 2 1
Explanation:
Traversing level 1 : 3 2
Traversing level 0 : 1
```

```
vector<int> reverseLevelOrder(Node *root)
{
    // code here
    vector<int> ans;


    if(root==NULL)
    return ans;

    queue<Node *>q;
    q.push(root);
    while(!q.empty())
    {
        Node *temp=q.front();
        q.pop();
        ans.push_back(temp->data);
        if(temp->right)
        q.push(temp->right);
        if(temp->left)
        q.push(temp->left);
    }
    reverse(ans.begin(),ans.end());
    return ans;
}
```

https://practice.geeksforgeeks.org/problems/reverse-level-order-traversal/1?page=1&category[]=Tree&curated[]=7&sortBy=submissions



ZigZag Tree Traversal

Easy   Accuracy: **49.78%**   Submissions: **84527**   Points: **2**

Given a Binary Tree. Find the Zig-Zag Level Order Traversal of the Binary Tree.

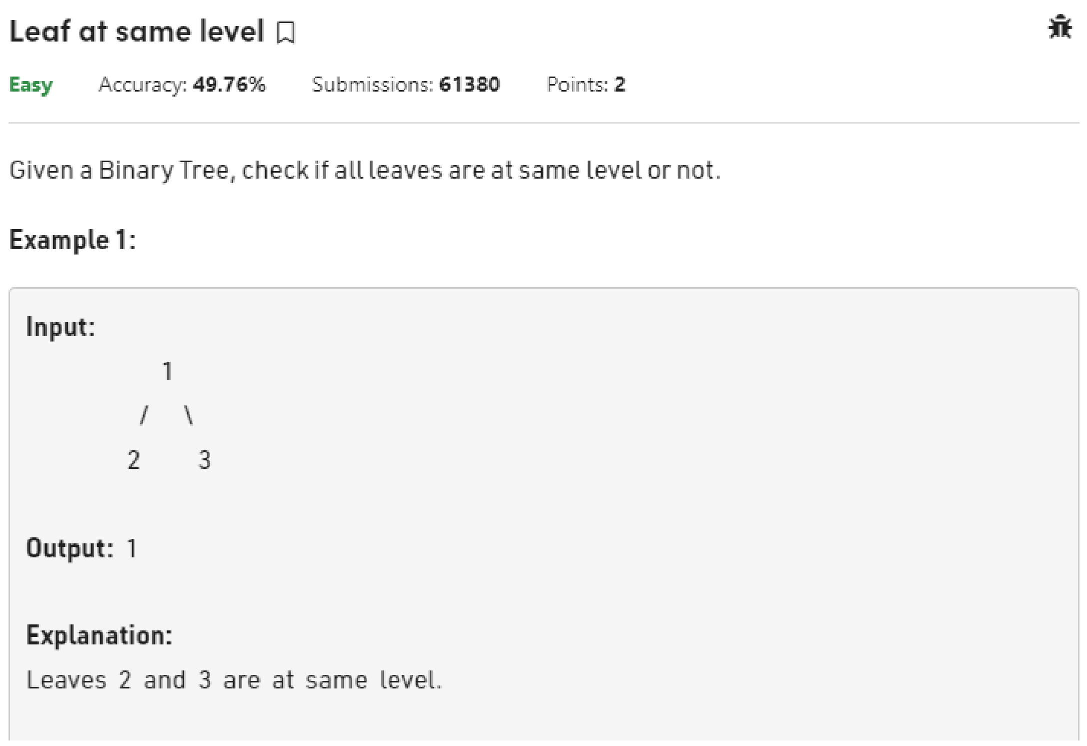
Example 1:

Input:
```
        1
      /   \
     2     3
    / \   /  \
   4   5 6    7
```
Output:
```
1 3 2 4 5 6 7
```

```
class Solution{
    public:
    //Function to store the zig zag order traversal of tree in a list.
    vector <int> zigZagTraversal(Node* root)
    {
      // Code here
      vector<int> result;
      queue<Node*> q;
      if(root==NULL)
      return result;

      bool lr=true;

      q.push(root);
      while(q.size()!=0)
      {

          //process level
          int sizee=q.size();

          vector<int> ans(sizee,0);
          //level process

          for(int i=0;i<sizee;i++)
          {
          Node *temp=q.front();
          q.pop();

          int index=lr?i:sizee-i-1;

          ans[index]=temp->data;

          if(temp->left)
          q.push(temp->left);
          if(temp->right)
          q.push(temp->right);

          }
          lr=!lr;
          for(auto i:ans){
              result.push_back(i);
          }
      }
```

```
        return result;

      }
  };
```

https://practice.geeksforgeeks.org/problems/zigzag-tree-traversal/1?page=1&category[]=Tree&curated[]=7&sortBy=submissions



Leaf at same level

Easy   Accuracy: **49.76%**   Submissions: **61380**   Points: **2**

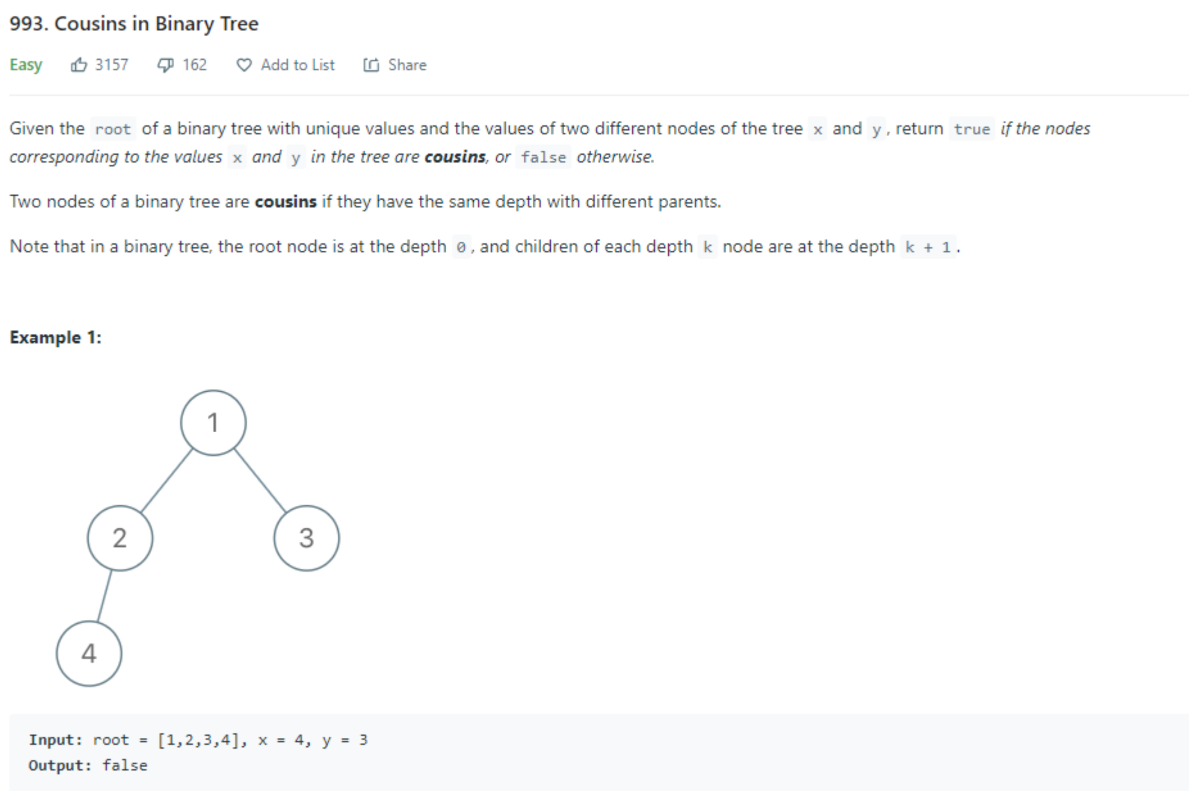
Given a Binary Tree, check if all leaves are at same level or not.

Example 1:

```
Input:
            1
          /   \
         2     3

Output: 1

Explanation:
Leaves 2 and 3 are at same level.
```

```
class Solution{
  public:
    /*You are required to complete this method*/

    bool check(Node *root)
    {
        //Your code here
       map<int,int> mp;
       queue<pair<Node*,int>> q;

       if(root==NULL)
       return false;

       q.push(make_pair(root,0));
       while(!q.empty())
       {
           pair<Node*,int> temp=q.front();
           q.pop();

           Node* r=temp.first;
           int level=temp.second;
           if(r->left)
           q.push(make_pair(r->left,level+1));
           if(r->right)
           q.push(make_pair(r->right,level+1));

           if(r->left==NULL && r->right==NULL)
           {
               mp[level]++;
           }
       }
       if(mp.size()==1)
       return true;

       return false;

    }
};
```

https://practice.geeksforgeeks.org/problems/leaf-at-same-level/1?page=1&category[]=Tree&curated[]=7&sortBy=submissions



993. Cousins in Binary Tree

Given the `root` of a binary tree with unique values and the values of two different nodes of the tree `x` and `y`, return `true` *if the nodes corresponding to the values* `x` *and* `y` *in the tree are **cousins**, or* `false` *otherwise.*

Two nodes of a binary tree are **cousins** if they have the same depth with different parents.

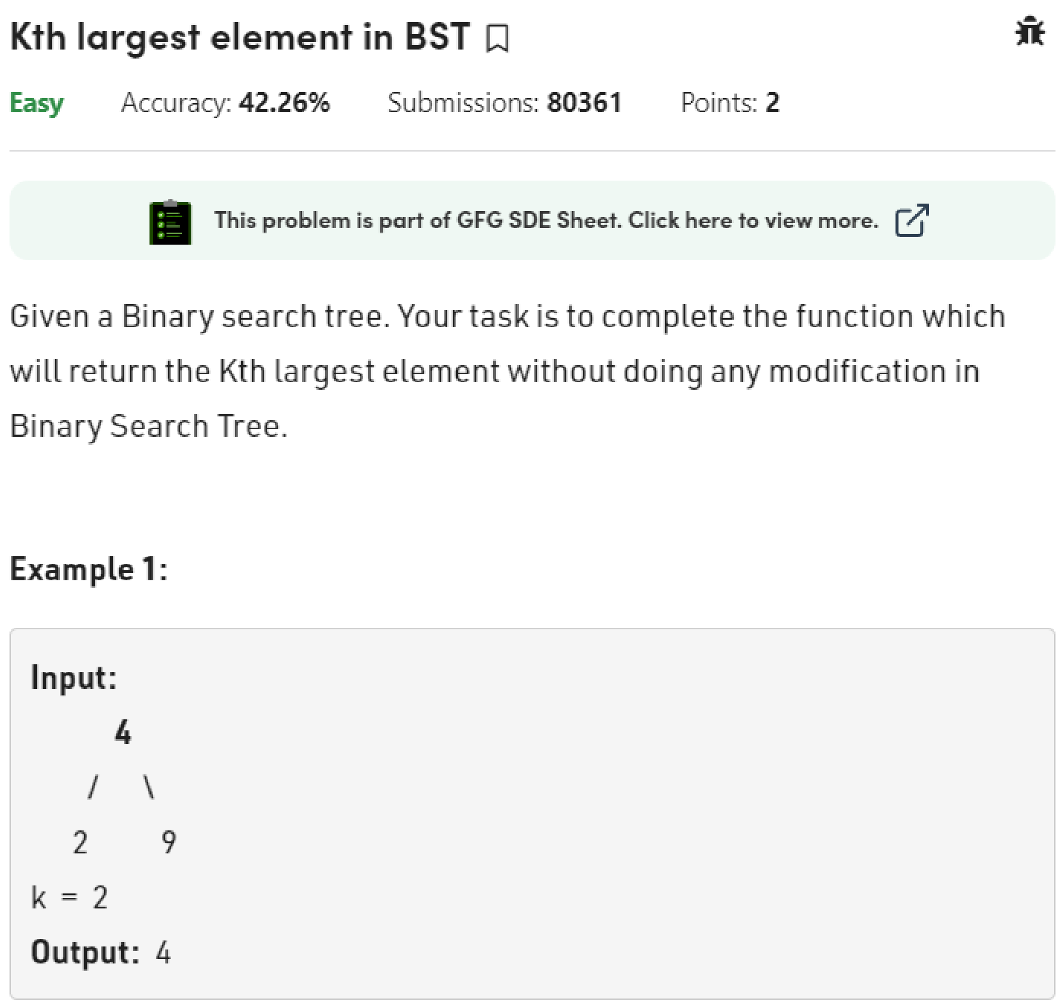
Note that in a binary tree, the root node is at the depth `0`, and children of each depth `k` node are at the depth `k + 1`.

Example 1:

```
Input: root = [1,2,3,4], x = 4, y = 3
Output: false
```

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:

    void solve(TreeNode *root,int x,int y,int level[],int parent[],int lev,int pare)
    {
        if(root==NULL)
            return;

        if(root->val==x)
        {
            level[0]=lev;
            parent[0]=pare;

        }
        if(root->val==y)
        {
            level[1]=lev;
            parent[1]=pare;

        }
        solve(root->left,x,y,level,parent,lev+1,root->val);
        solve(root->right,x,y,level,parent,lev+1,root->val);


    }
    bool isCousins(TreeNode* root, int x, int y) {

        int level[2]={-1,-1};
        int parent[2]={-1,-1};

        solve(root,x,y,level,parent,0,root->val);
        int lev=0;
        if(level[0]==level[1]&& parent[0]!=parent[1])
            return true;

        return false;

    }
};
```

https://leetcode.com/problems/cousins-in-binary-tree/



Kth largest element in BST

Easy Accuracy: **42.26%** Submissions: **80361** Points: **2**

This problem is part of GFG SDE Sheet. Click here to view more.

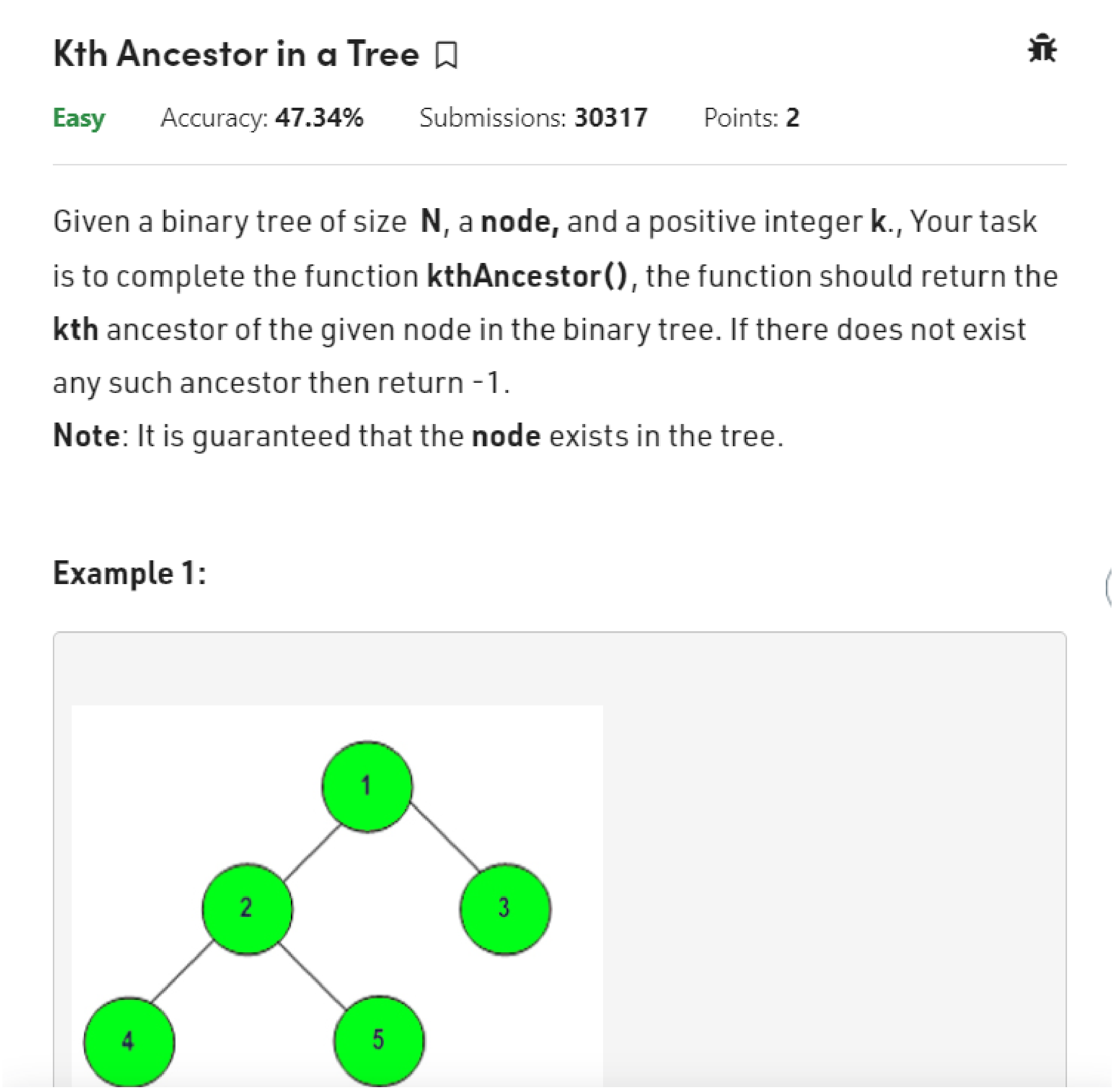
Given a Binary search tree. Your task is to complete the function which will return the Kth largest element without doing any modification in Binary Search Tree.

Example 1:

```
Input:
      4
    /   \
   2     9
k = 2
Output: 4
```

```
class Solution
{
    public:
    void solve(Node *root,vector<int> &ans)
    {
        if(root==NULL)
        return;

        ans.push_back(root->data);
        solve(root->left,ans);
        solve(root->right,ans);
    }
    int kthLargest(Node *root, int K)
    {
        //Your code here
        if(root==NULL)
        return 0;
        vector<int> ans;
        solve(root,ans);
        sort(ans.begin(),ans.end());
        int size=ans.size();
        return ans[size-K];
    }
};
```

https://practice.geeksforgeeks.org/problems/kth-largest-element-in-bst/1?page=2&category[]=Tree&sortBy=submissions



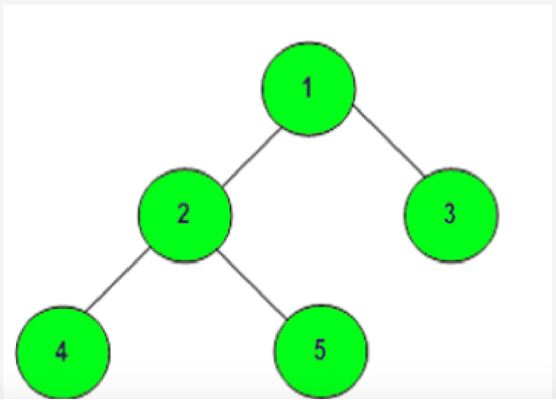
Kth Ancestor in a Tree

Easy Accuracy: **47.34%** Submissions: **30317** Points: **2**

Given a binary tree of size **N**, a **node,** and a positive integer **k**., Your task is to complete the function **kthAncestor()**, the function should return the **kth** ancestor of the given node in the binary tree. If there does not exist any such ancestor then return -1.

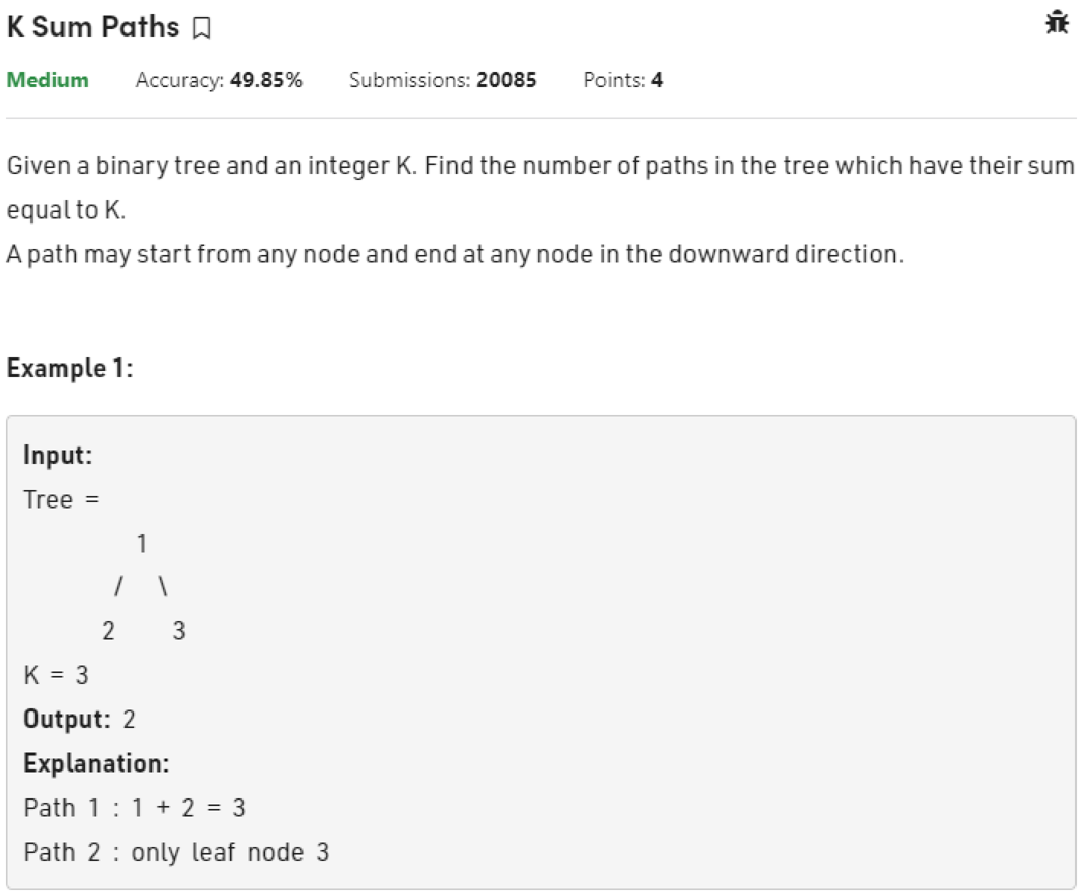
Note: It is guaranteed that the **node** exists in the tree.

Example 1:



```
bool solve(Node *root,int node,vector<int> &path)
{
    if(root==NULL)
    return false;

    path.push_back(root->data);

    bool leftans=solve(root->left,node,path);
    bool rightans=solve(root->right,node,path);

    if(root->data==node)
    return true;

    if(leftans||rightans)
    return true;

    path.pop_back();
    return false;


}
int kthAncestor(Node *root, int k, int node)
{
    // Code here
    if(root->data==node)
    return -1;
    vector<int> path;
    bool t=solve(root,node,path);
    int size=path.size();
    if(k>=size)
    return -1;
    return path[size-k-1];

}
```

https://practice.geeksforgeeks.org/problems/kth-ancestor-in-a-tree/1?page=2&category[]=Tree&curated[]=7&sortBy=submissions



K Sum Paths

Medium Accuracy: 49.85% Submissions: 20085 Points: 4

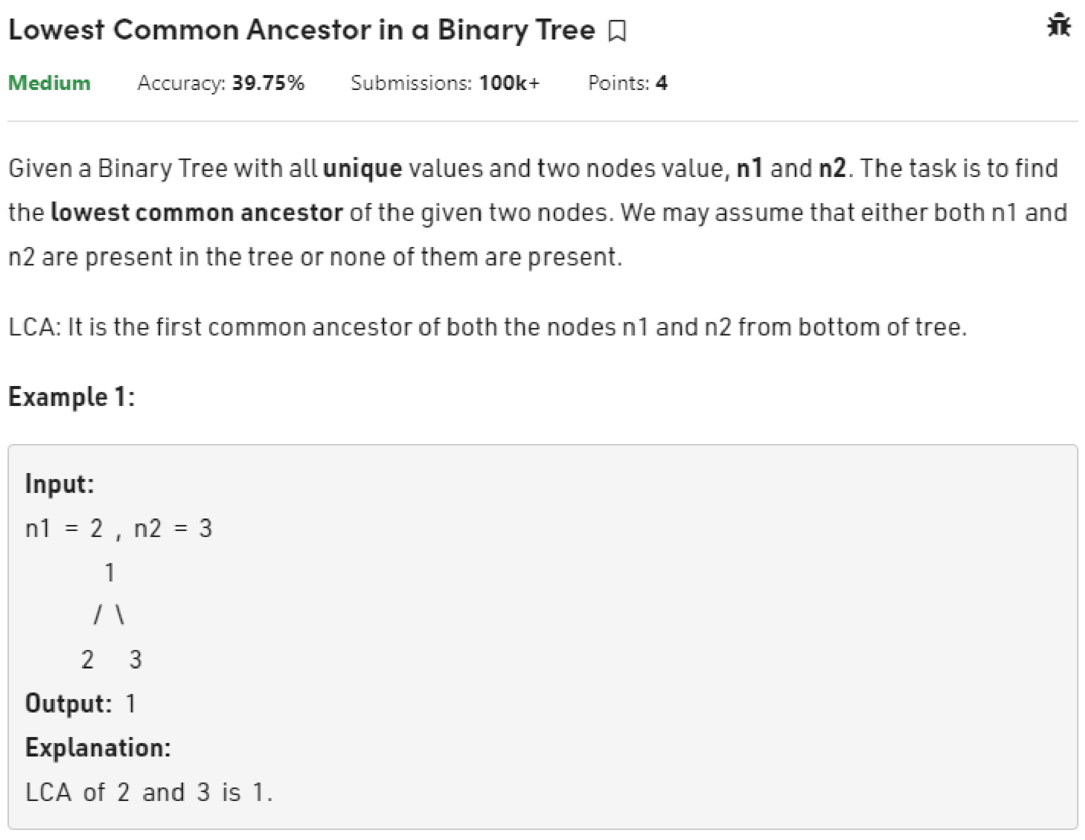
Given a binary tree and an integer K. Find the number of paths in the tree which have their sum equal to K.
A path may start from any node and end at any node in the downward direction.

Example 1:

```
Input:
Tree =
          1
        /   \
       2     3
K = 3
Output: 2
Explanation:
Path 1 : 1 + 2 = 3
Path 2 : only leaf node 3
```

```
class Solution{
  public:
    void solve(Node *root,int k,int &count,vector<int> path)
    {
        if(root==NULL)
        return;

        path.push_back(root->data);

        solve(root->left,k,count,path);
        solve(root->right,k,count,path);

        //check for sum
        int sum=0;
        int size=path.size();
        for(int i=size-1;i>=0;i--)
        {
            sum+=path[i];
            if(sum==k)
            count++;

        }
        path.pop_back();

    }
    int sumK(Node *root,int k)
    {
        // code here
        vector<int> path;
        int count=0;
        solve(root,k,count,path);

        return count;
    }
};
```

https://practice.geeksforgeeks.org/problems/k-sum-paths/1?page=3&category[]=Tree&curated[]=7&sortBy=submissions



Lowest Common Ancestor in a Binary Tree

Medium Accuracy: **39.75%** Submissions: **100k+** Points: **4**

Given a Binary Tree with all **unique** values and two nodes value, **n1** and **n2**. The task is to find the **lowest common ancestor** of the given two nodes. We may assume that either both n1 and n2 are present in the tree or none of them are present.

LCA: It is the first common ancestor of both the nodes n1 and n2 from bottom of tree.

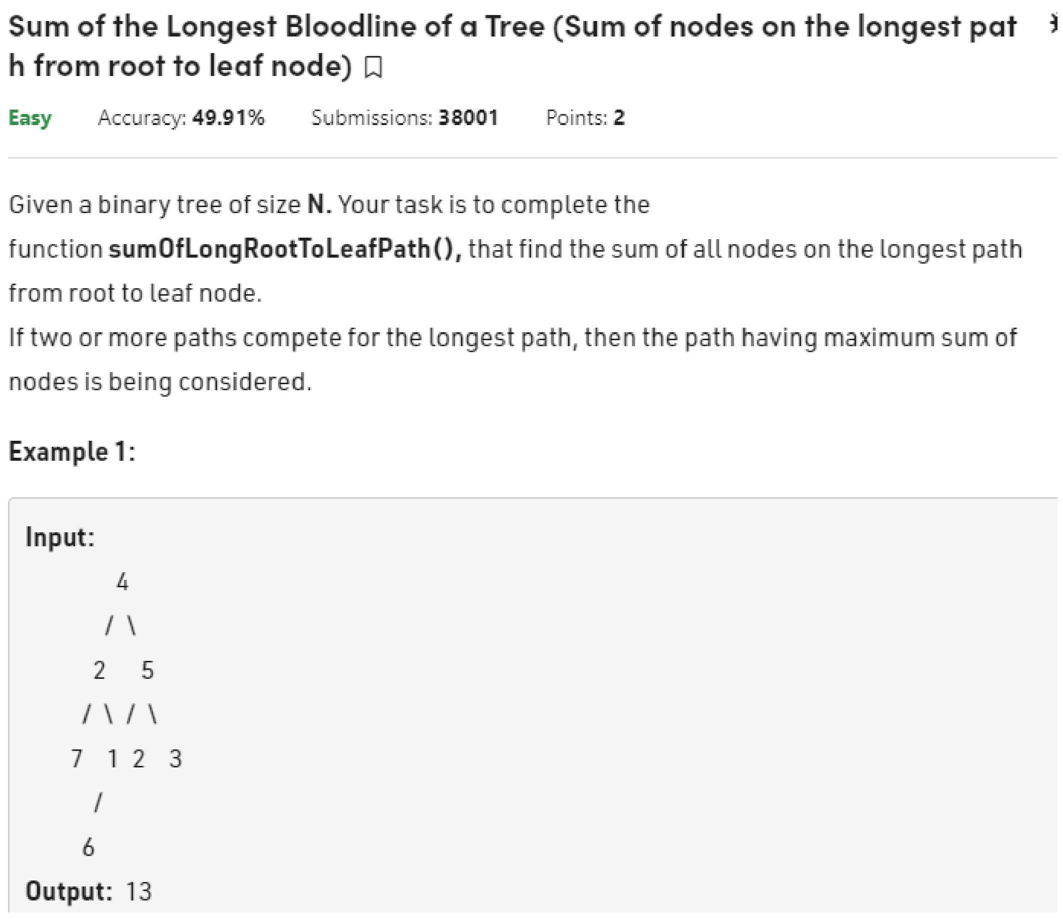
Example 1:

```
Input:
n1 = 2 , n2 = 3
       1
      / \
    2    3
Output: 1
Explanation:
LCA of 2 and 3 is 1.
```

```
class Solution
{
    public:
    //Function to return the lowest common ancestor in a Binary Tree.
    Node* lca(Node* root ,int n1 ,int n2 )
    {
       //Your code here
       if(root==NULL)
       return NULL;

       if(root->data==n1 ||root->data==n2)
       {
           return root;
       }

       Node *leftans=lca(root->left,n1,n2);
       Node *rightans=lca(root->right,n1,n2);

       if(leftans!=NULL && rightans!=NULL)
```

```
        return root;
        else if(rightans!=NULL && leftans==NULL)
        return rightans;
        else if(leftans!=NULL && rightans==NULL)
        return leftans;

    }
};
```

https://practice.geeksforgeeks.org/problems/lowest-common-ancestor-in-a-binary-tree/1?page=2&category[]=Tree&curated[]=7&sortBy=submissions



Sum of the Longest Bloodline of a Tree (Sum of nodes on the longest pat h from root to leaf node)

Easy Accuracy: **49.91%** Submissions: **38001** Points: **2**

Given a binary tree of size **N**. Your task is to complete the function **sumOfLongRootToLeafPath(),** that find the sum of all nodes on the longest path from root to leaf node.
If two or more paths compete for the longest path, then the path having maximum sum of nodes is being considered.

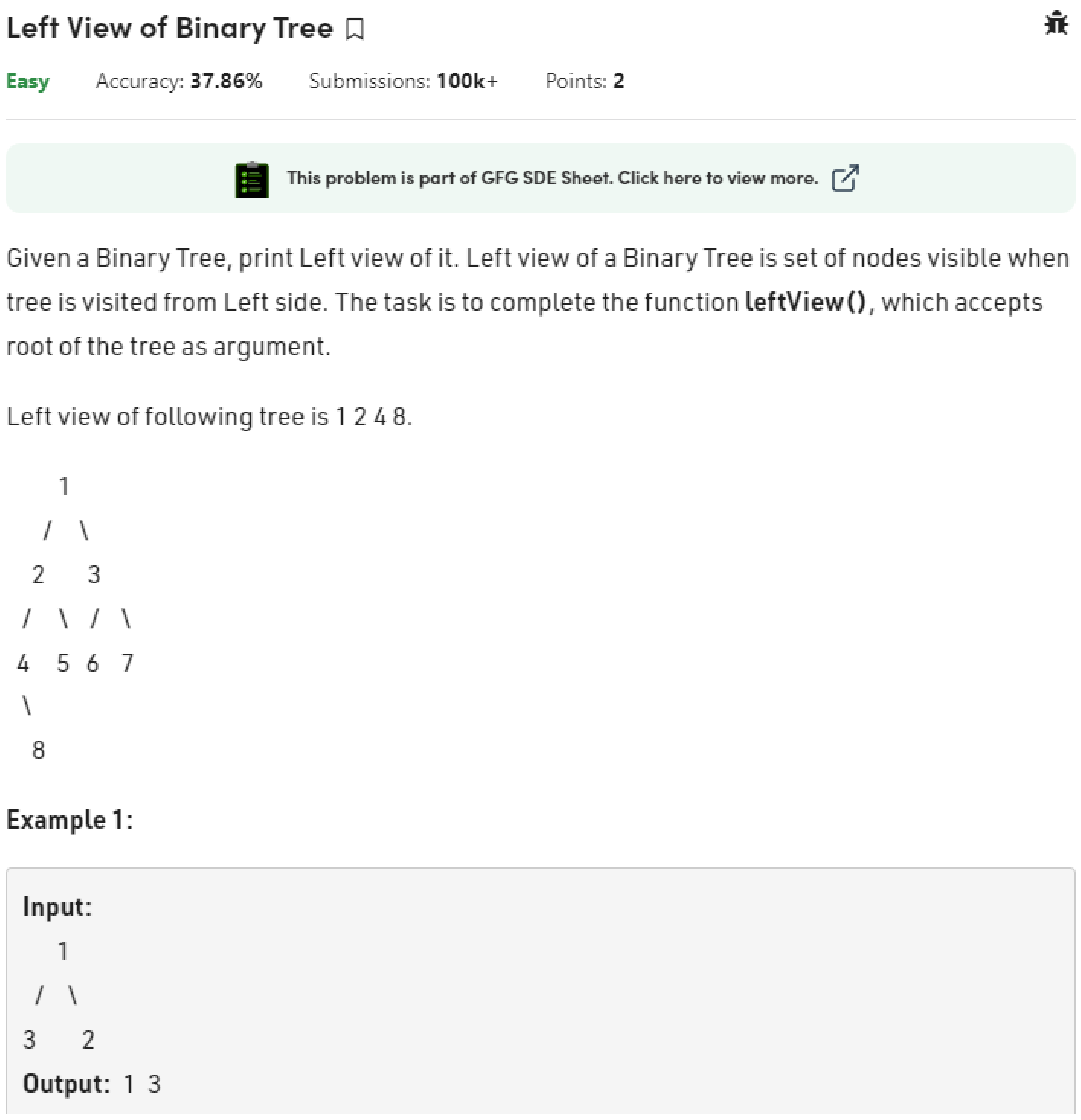
Example 1:

```
Input:
        4
       / \
      2   5
     / \ / \
    7  1 2  3
      /
     6
Output: 13
```

```
class Solution
{
public:

    void solve(Node *root,int len,int &maxlen,int sum,int &maxsum)
    {
        if(root==NULL)
       {
           if(maxlen<len)
           {
               maxsum=sum;
               maxlen=len;

           }
           else if(len==maxlen)
           {
               maxsum=max(maxsum,sum);
           }
           return;
       }
```

```
        sum=sum+root->data;

        solve(root->left,len+1,maxlen,sum,maxsum);
        solve(root->right,len+1,maxlen,sum,maxsum);
    }

    int sumOfLongRootToLeafPath(Node *root)
    {
        //code here
        int len=0;
        int maxlen=0;
        int sum=0;
        int maxsum=INT_MIN;
        solve(root,len,maxlen,sum,maxsum);
        return maxsum;
    }
};
```

https://practice.geeksforgeeks.org/problems/sum-of-the-longest-bloodline-of-a-tree/1?page=2&category[]=Tree&curated[]=7&sortBy=submissions



Left View of Binary Tree

Easy Accuracy: **37.86%** Submissions: **100k+** Points: **2**

This problem is part of GFG SDE Sheet. Click here to view more.

Given a Binary Tree, print Left view of it. Left view of a Binary Tree is set of nodes visible when tree is visited from Left side. The task is to complete the function **leftView()**, which accepts root of the tree as argument.

Left view of following tree is 1 2 4 8.

```
      1
    /   \
   2     3
  / \   / \
 4   5 6   7
  \
   8
```

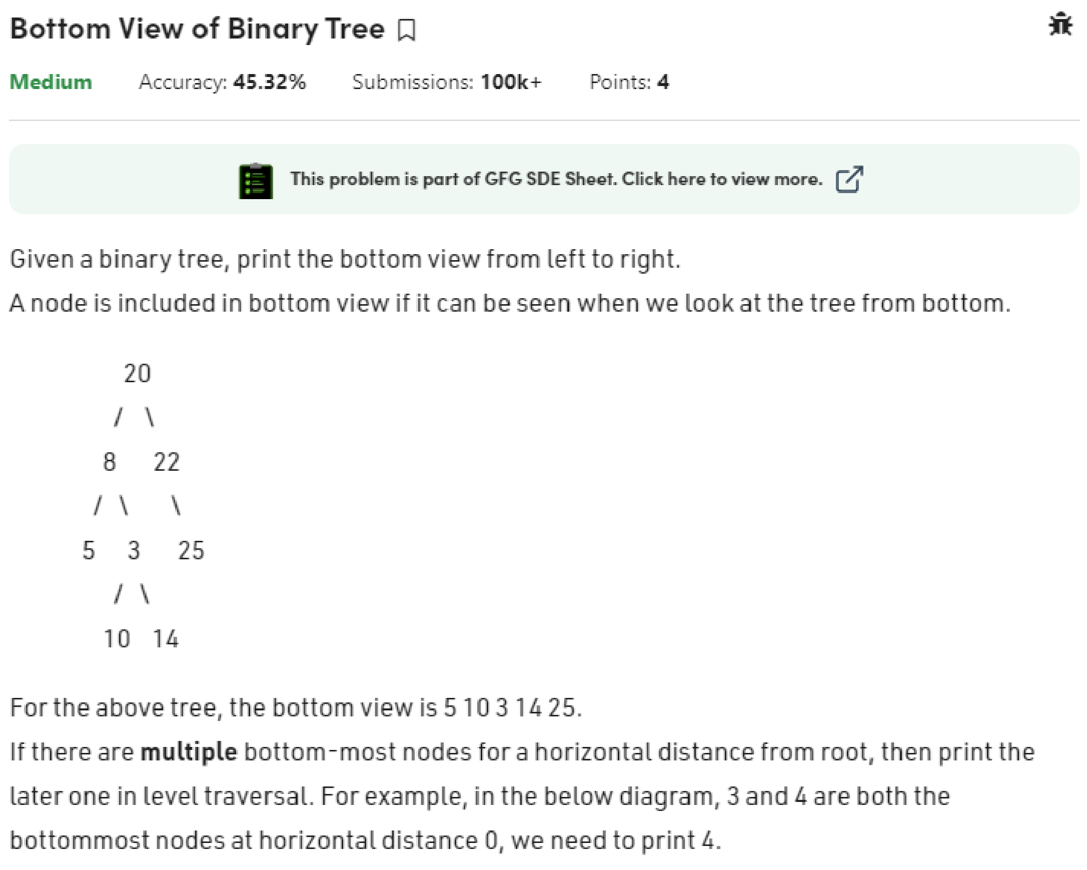
Example 1:

Input:
```
   1
 /  \
3    2
```
Output: 1 3

```
//Function to return a list containing elements of left view of the binary tree.
void getleft(Node* root,vector<int> &ans,int lev)
{
    if(root==NULL)
```

```
      return;

      if(lev==ans.size())
      {
          ans.push_back(root->data);

      }




      getleft(root->left,ans,lev+1);
      getleft(root->right,ans,lev+1);


  }
  vector<int> leftView(Node *root)
  {
     // Your code here
     vector<int> ans;
     int lev=0;
     getleft(root,ans,lev);

     return ans;

  }
```

https://practice.geeksforgeeks.org/problems/left-view-of-binary-tree/1?page=1&category[]=Tree&curated[]=7&sortBy=submissions



Bottom View of Binary Tree

Medium Accuracy: 45.32% Submissions: 100k+ Points: 4

This problem is part of GFG SDE Sheet. Click here to view more.

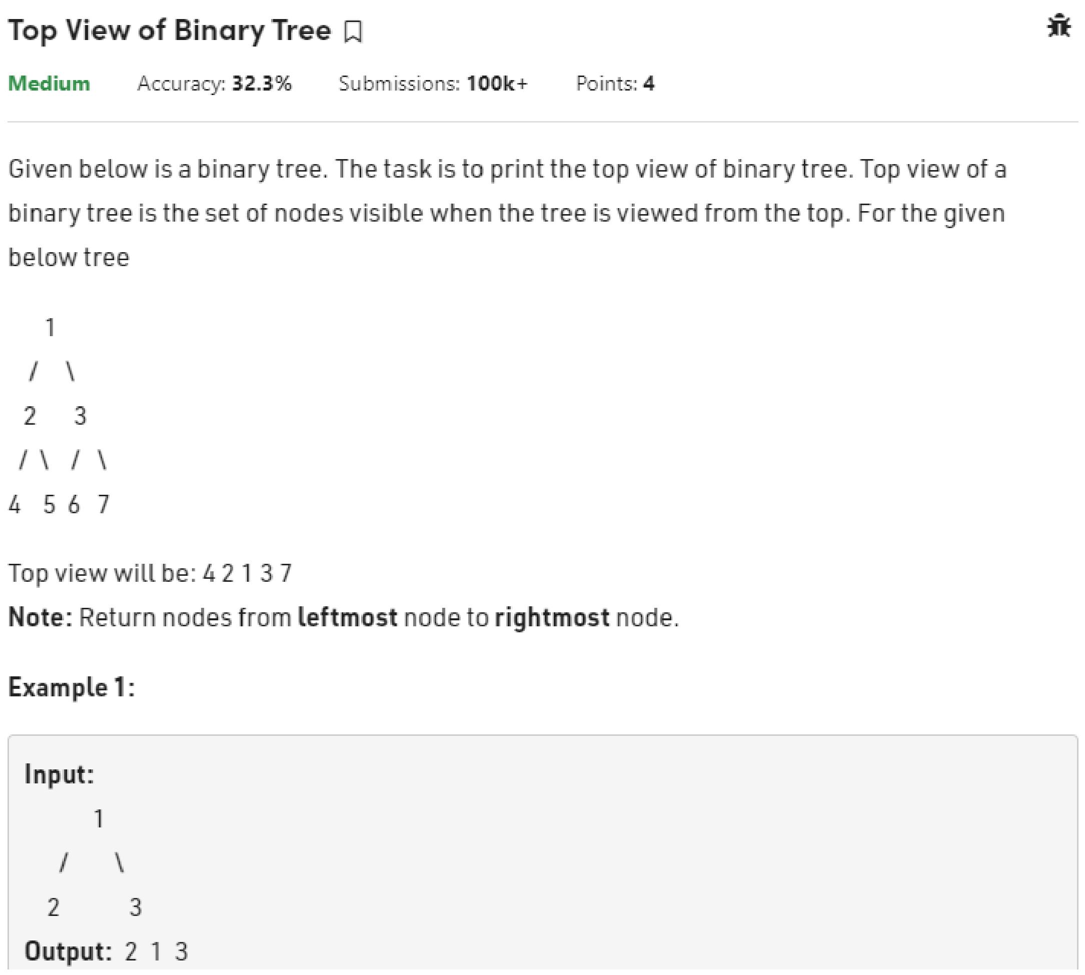
Given a binary tree, print the bottom view from left to right.
A node is included in bottom view if it can be seen when we look at the tree from bottom.

```
          20
         /  \
       8     22
      / \      \
    5    3      25
        / \
      10   14
```

For the above tree, the bottom view is 5 10 3 14 25.
If there are **multiple** bottom-most nodes for a horizontal distance from root, then print the later one in level traversal. For example, in the below diagram, 3 and 4 are both the bottommost nodes at horizontal distance 0, we need to print 4.

```
class Solution {
  public:
    vector <int> bottomView(Node *root) {
        // Your Code Here
        //Your code here
        vector<int> result;
        if(root==NULL)
        return result;
        map<int,int> topnode;
        queue<pair<Node*,int>> q;
        q.push(make_pair(root,0));

        while(!q.empty())
        {
            pair<Node*,int> temp=q.front();
            q.pop();
            Node *frontnode=temp.first;
            int dist=temp.second;

            topnode[dist]=(frontnode->data);

            if(frontnode->left)
            {
                q.push(make_pair(frontnode->left,dist-1));

            }
            if(frontnode->right)
            {
                q.push(make_pair(frontnode->right,dist+1));
            }


        }
        for(auto i:topnode)
        {
            result.push_back(i.second);
        }
        return result;
    }
};
```

https://practice.geeksforgeeks.org/problems/bottom-view-of-binary-tree/1?page=2&category[]=Tree&curated[]=7&sortBy=submissions



Top View of Binary Tree

Medium Accuracy: **32.3%** Submissions: **100k+** Points: **4**

Given below is a binary tree. The task is to print the top view of binary tree. Top view of a binary tree is the set of nodes visible when the tree is viewed from the top. For the given below tree

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

Top view will be: 4 2 1 3 7

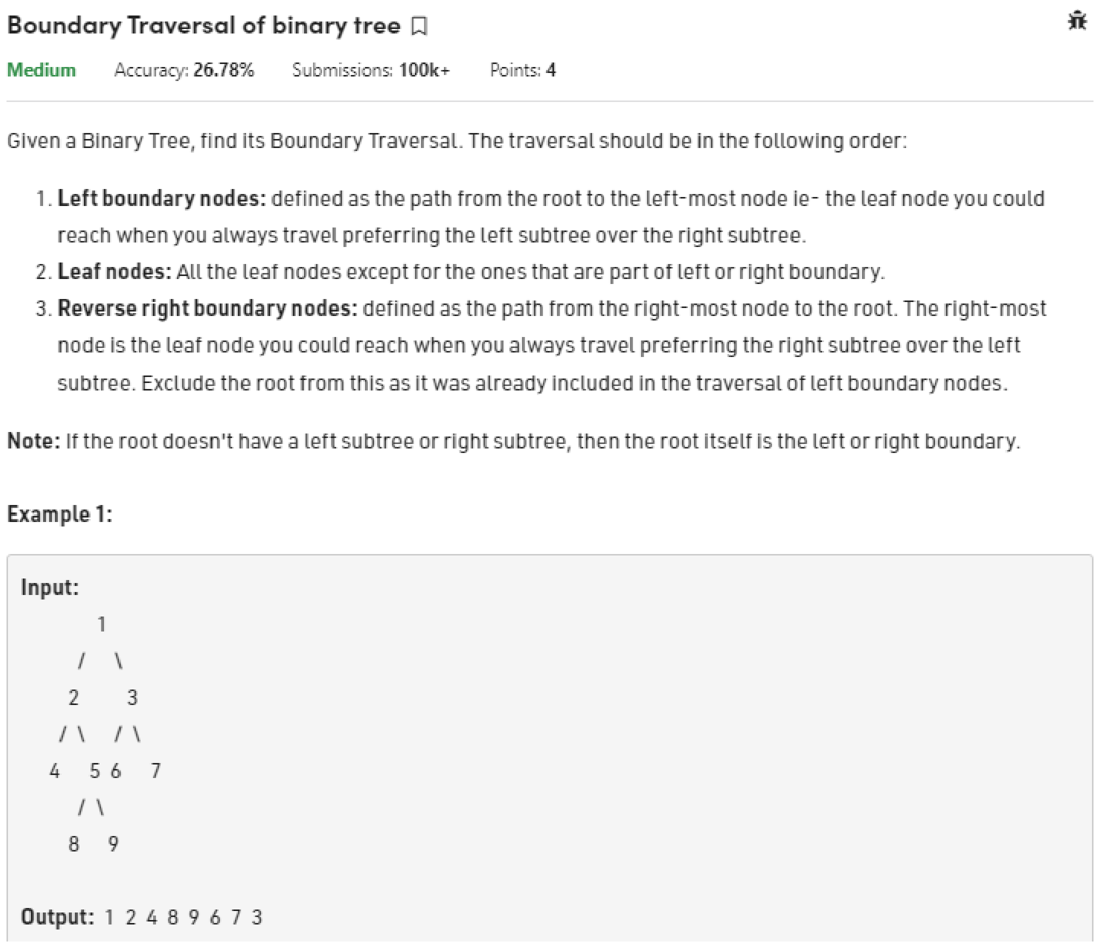
Note: Return nodes from **leftmost** node to **rightmost** node.

Example 1:

Input:
```
       1
    /     \
   2       3
```
Output: 2 1 3

```
class Solution
{
    public:
    //Function to return a list of nodes visible from the top view
    //from left to right in Binary Tree.
    vector<int> topView(Node *root)
    {
        //Your code here
        vector<int> result;
        if(root==NULL)
        return result;
        map<int,int> topnode;
        queue<pair<Node*,int>> q;
        q.push(make_pair(root,0));

        while(!q.empty())
        {
            pair<Node*,int> temp=q.front();
            q.pop();
            Node *frontnode=temp.first;
            int dist=temp.second;
            if(topnode.find(dist)==topnode.end())
            topnode[dist]=(frontnode->data);

            if(frontnode->left)
            {
                q.push(make_pair(frontnode->left,dist-1));

            }
            if(frontnode->right)
            {
                q.push(make_pair(frontnode->right,dist+1));
            }
        }
        for(auto i:topnode)
        {
            result.push_back(i.second);
        }
        return result;
    }

};
```

https://practice.geeksforgeeks.org/problems/top-view-of-binary-tree/1?page=2&category[]=Tree&curated[]=7&sortBy=submissions



Boundary Traversal of binary tree

Medium   Accuracy: **26.78%**   Submissions: **100k+**   Points: **4**

Given a Binary Tree, find its Boundary Traversal. The traversal should be in the following order:

1. **Left boundary nodes:** defined as the path from the root to the left-most node ie- the leaf node you could reach when you always travel preferring the left subtree over the right subtree.
2. **Leaf nodes:** All the leaf nodes except for the ones that are part of left or right boundary.
3. **Reverse right boundary nodes:** defined as the path from the right-most node to the root. The right-most node is the leaf node you could reach when you always travel preferring the right subtree over the left subtree. Exclude the root from this as it was already included in the traversal of left boundary nodes.

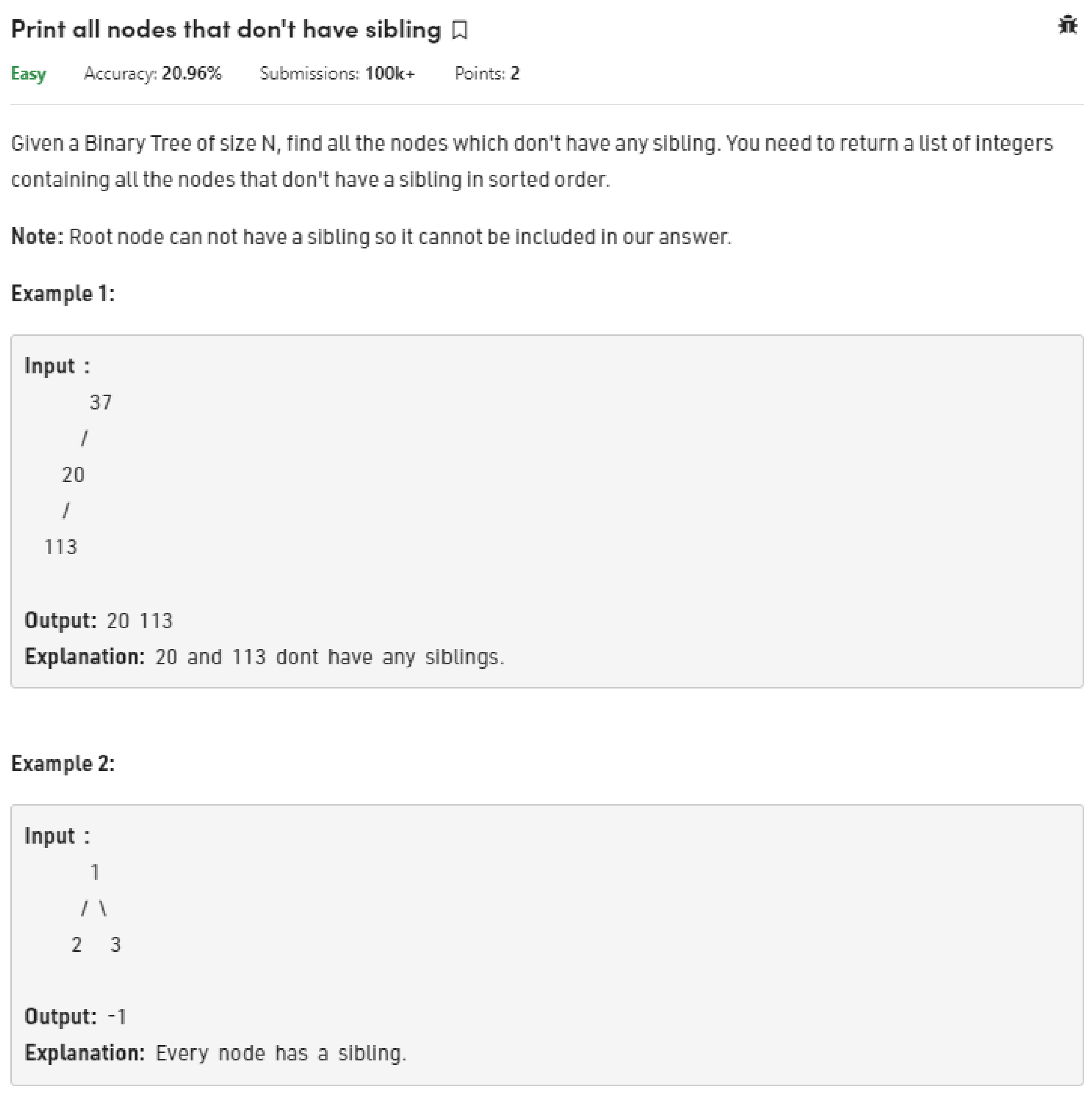
Note: If the root doesn't have a left subtree or right subtree, then the root itself is the left or right boundary.

Example 1:

```
Input:
        1 
      /   \
     2     3  
    / \   / \ 
   4   5 6   7
      / \
     8   9
   
Output: 1 2 4 8 9 6 7 3
```

```
class Solution {
public:
    void leftt(Node *root,vector<int> &ans)
    {
        if(root==NULL)
        return;

        if(root->left==NULL && root->right==NULL)
        return;

        ans.push_back(root->data);
        if(root->left)
        leftt(root->left,ans);
        else
        leftt(root->right,ans);
    }
    void leaf(Node *root,vector<int> &ans)
    {
        if(root==NULL)
        return;
        if(root->left==NULL && root->right==NULL)
        {
        ans.push_back(root->data);
        return;
        }
        leaf(root->left,ans);
        leaf(root->right,ans);
    }
    void rightt(Node *root,vector<int> &ans)
    {
        if(root==NULL)
        {
            return;
        }
        if(root->left==NULL && root->right==NULL)
        return ;


        if(root->right)
        rightt(root->right,ans);
        else
        rightt(root->left,ans);

        ans.push_back(root->data);

    }
    vector <int> boundary(Node *root)
    {
        //Your code here
        vector<int> ans;

        if(root==NULL)
        return ans;

        ans.push_back(root->data);
        leftt(root->left,ans);
        leaf(root->left,ans);
        leaf(root->right,ans);

        rightt(root->right,ans);

        return ans;
    }
};
```

https://practice.geeksforgeeks.org/problems/boundary-traversal-of-binary-tree/1?page=2&category[]=Tree&curated[]=7&sortBy=submissions



Print all nodes that don't have sibling

Easy   Accuracy: **20.96%**   Submissions: **100k+**   Points: **2**

Given a Binary Tree of size N, find all the nodes which don't have any sibling. You need to return a list of integers containing all the nodes that don't have a sibling in sorted order.

Note: Root node can not have a sibling so it cannot be included in our answer.

Example 1:

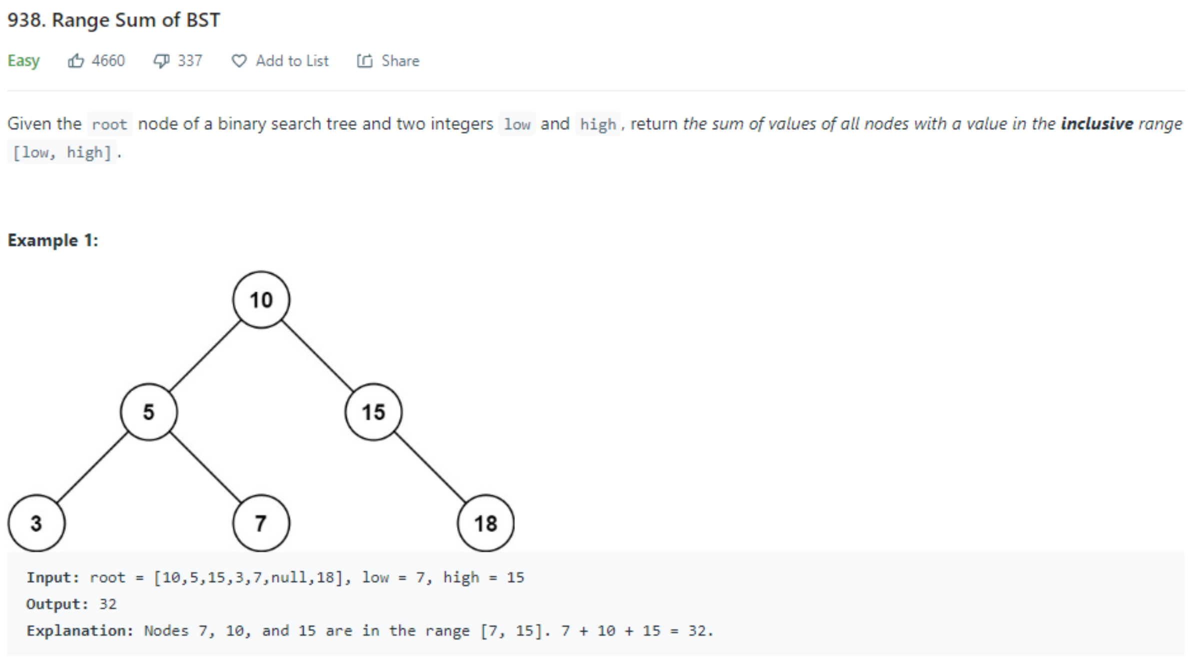
```
Input :
       37
      /
    20
    /
  113

Output: 20 113
Explanation: 20 and 113 dont have any siblings.
```

Example 2:

```
Input :
       1
      / \
     2   3

Output: -1
Explanation: Every node has a sibling.
```

```
void solve(Node *root,vector<int> &ans)
{
    if(root==NULL)
    return;

    if(root->left==NULL && root->right!=NULL)
    {
    ans.push_back(root->right->data);

    }
    else if(root->left!=NULL && root->right==NULL)
    {
    ans.push_back(root->left->data);

    }

    solve(root->left,ans);
    solve(root->right,ans);

    return;

}
vector<int> noSibling(Node* node)
{
    // code here

    vector<int> ans;
    if(node==NULL)
    {
        ans.push_back(-1);
        return ans;
    }

    solve(node,ans);

    if(ans.size()==0)
    {
        ans.push_back(-1);

    }
    sort(ans.begin(),ans.end());

    return ans;

}
```

https://practice.geeksforgeeks.org/problems/print-all-nodes-that-dont-have-sibling/1



938. Range Sum of BST

Easy 4660 337 Add to List Share

Given the `root` node of a binary search tree and two integers `low` and `high`, return *the sum of values of all nodes with a value in the* ***inclusive*** *range* `[low, high]`.

Example 1:

```
Input: root = [10,5,15,3,7,null,18], low = 7, high = 15
Output: 32
Explanation: Nodes 7, 10, and 15 are in the range [7, 15]. 7 + 10 + 15 = 32.
```

```
class Solution {
public:
    int rangeSumBST(TreeNode* root, int low, int high) {
        if(root==NULL)
            return 0;


        int data=root->val;
        if(data>=low && data<=high)
        {
            return data+rangeSumBST(root->left,low,high)+rangeSumBST(root->right,low,high);
        }
        return rangeSumBST(root->left,low,high)+rangeSumBST(root->right,low,high);
    }
};
```

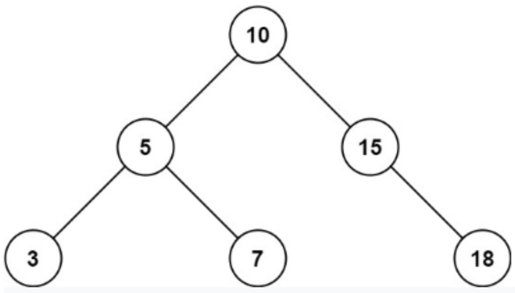
https://leetcode.com/problems/range-sum-of-bst/

938. Range Sum of BST

Easy 4660 337 Add to List Share

Given the `root` node of a binary search tree and two integers `low` and `high`, return *the sum of values of all nodes with a value in the **inclusive** range* `[low, high]`.

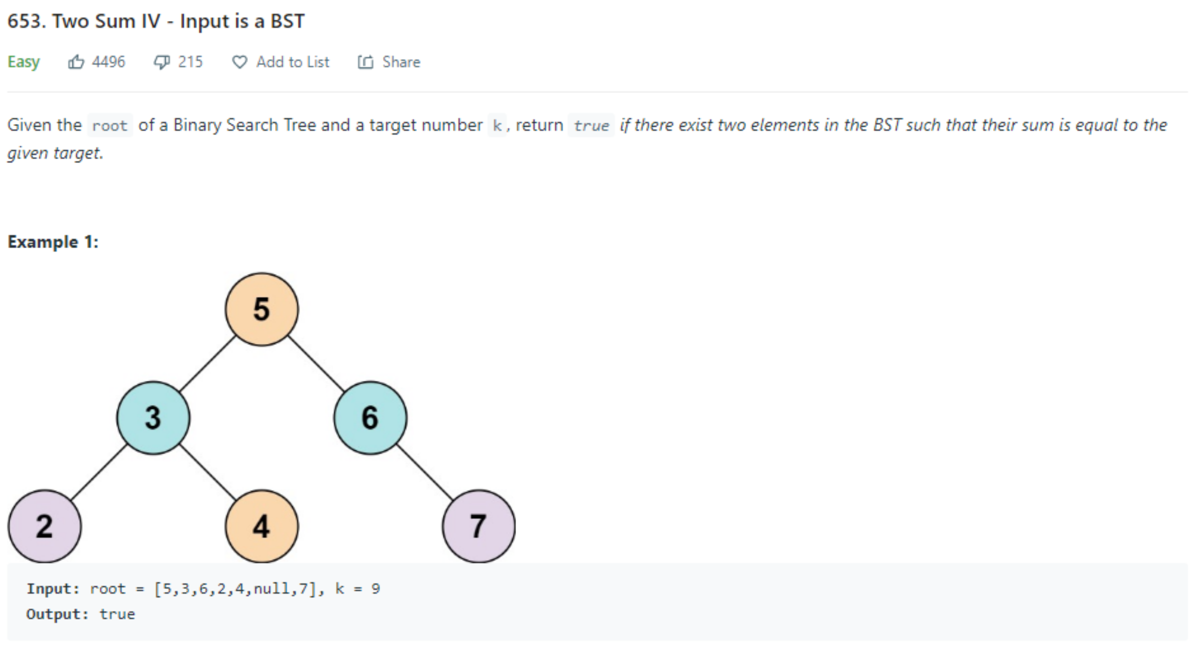
Example 1:



```
Input: root = [10,5,15,3,7,null,18], low = 7, high = 15
Output: 32
Explanation: Nodes 7, 10, and 15 are in the range [7, 15]. 7 + 10 + 15 = 32.
```

```
class Solution {
public:
    void getinor(TreeNode *root,vector<int> &ans)
    {
        if(root==NULL)
            return;

        ans.push_back(root->val);
        getinor(root->left,ans);
        getinor(root->right,ans);
    }
    TreeNode* increasingBST(TreeNode* root) {
        if(root==NULL)
            return NULL;

        vector<int> ans;
        getinor(root,ans);
        sort(ans.begin(),ans.end());
        TreeNode *newroot=new TreeNode(ans[0]);
        TreeNode *dumy=newroot;
        for(int i=1;i<ans.size();i++)
        {
            newroot->right=new TreeNode(ans[i]);
            newroot=newroot->right;
        }
        return dumy;

    }
};
```

https://leetcode.com/problems/range-sum-of-bst/

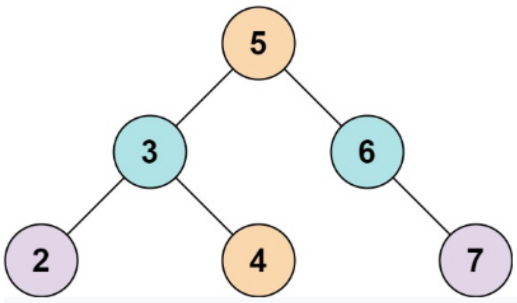


653. Two Sum IV - Input is a BST

Easy 4496 215 Add to List Share

Given the `root` of a Binary Search Tree and a target number `k`, return `true` *if there exist two elements in the BST such that their sum is equal to the given target.*

Example 1:

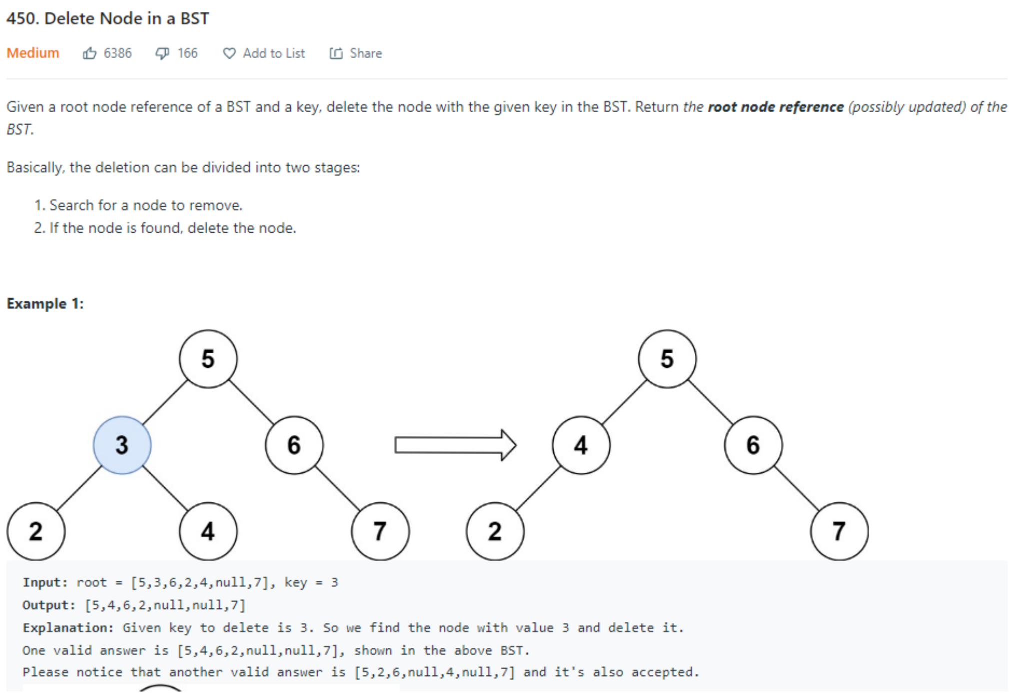


```
Input: root = [5,3,6,2,4,null,7], k = 9
Output: true
```

```
INEFFICIENT-->
class Solution {
public:
    void getino(TreeNode* root,vector<int> &ans)
    {
        if(root==NULL)
            return;

        ans.push_back(root->val);

        getino(root->left,ans);
        getino(root->right,ans);
    }
    bool findTarget(TreeNode* root, int k) {
        if(root==NULL)
            return false;
        vector<int> ans;
        getino(root,ans);
        sort(ans.begin(),ans.end());
        int flag=0;
        for(int i=0;i<ans.size();i++)
        {
            if(find(ans.begin(),ans.end(),k-ans[i])!=ans.end()&& find(ans.begin(),ans.end(),k-ans[i])!=ans.begin()+i)
            {
                flag++;
            }
        }
        if(flag==0)
            return false;

        return true;
    }
};



EFFICIENT--> (BINARY SEARCH)
class Solution {
public:
    void getino(TreeNode* root,vector<int> &ans)
    {
        if(root==NULL)
            return;
        ans.push_back(root->val);

        getino(root->left,ans);
        getino(root->right,ans);
    }
    bool findTarget(TreeNode* root, int k) {
        if(root==NULL)
            return false;
        vector<int> ans;
        getino(root,ans);
        sort(ans.begin(),ans.end());
        int low=0;
        int high=ans.size()-1;
        while(low<high)
        {
            if(ans[low]+ans[high]==k)
                return true;
            else if(ans[low]+ans[high]>k)
                high--;
            else
                low++;
        }
        return false;
    }
};
```

https://leetcode.com/problems/two-sum-iv-input-is-a-bst/submissions/

Important—>


450. Delete Node in a BST

Medium 6386 166 Add to List Share

Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return *the **root node reference** (possibly updated) of the BST*.

Basically, the deletion can be divided into two stages:

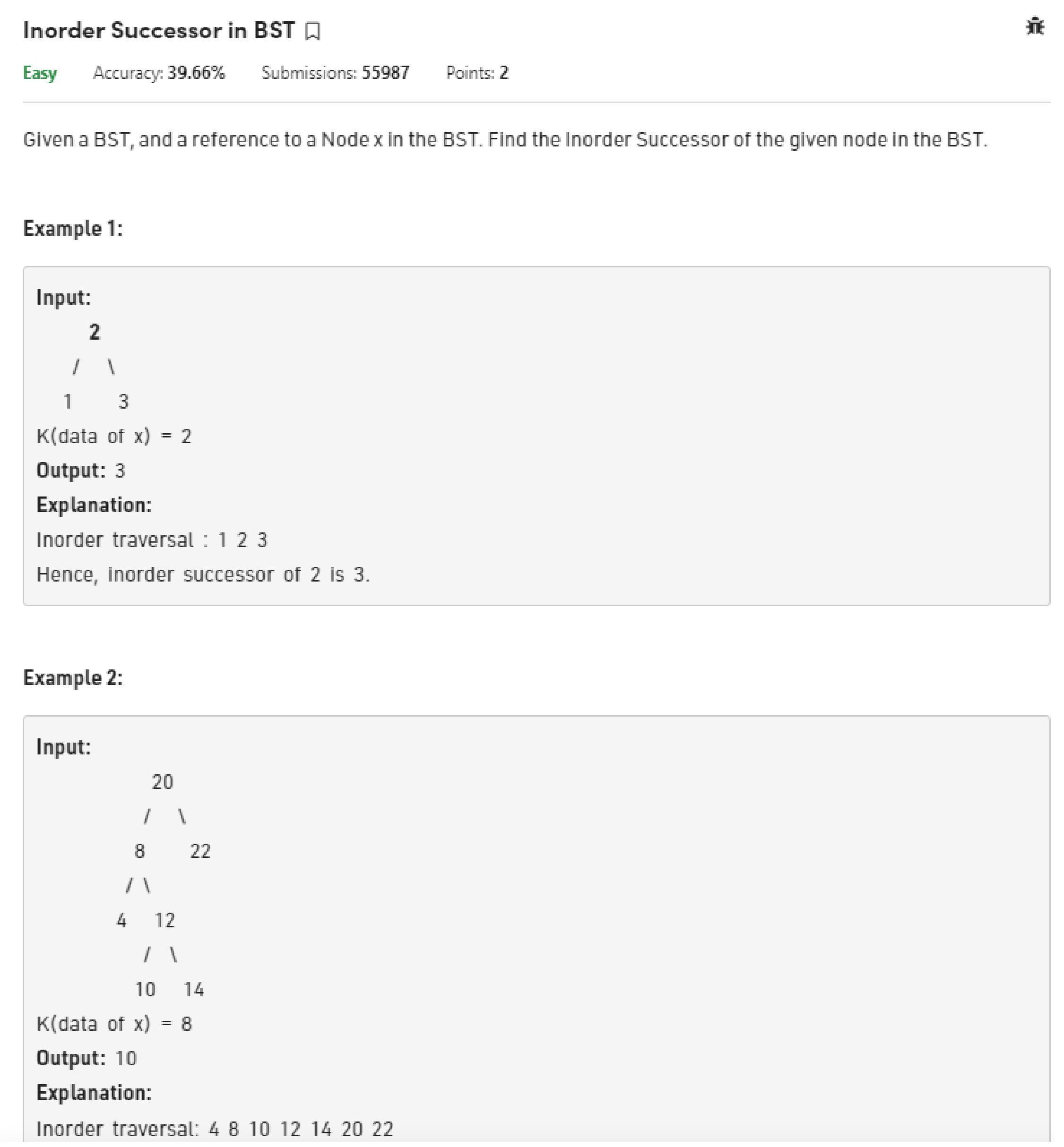
1. Search for a node to remove.
2. If the node is found, delete the node.

Example 1:

```
Input: root = [5,3,6,2,4,null,7], key = 3
Output: [5,4,6,2,null,null,7]
Explanation: Given key to delete is 3. So we find the node with value 3 and delete it.
One valid answer is [5,4,6,2,null,null,7], shown in the above BST.
Please notice that another valid answer is [5,2,6,null,4,null,7] and it's also accepted.
```

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
   TreeNode *lefty(TreeNode *root)
   {
       while(root->left!=NULL)
       {
           root=root->left;
       }
       return root;
   }
    TreeNode* deleteNode(TreeNode* root, int key) {
        if(root==NULL)
            return NULL;


        if(root->val>key)
        {
            root->left=deleteNode(root->left,key);
        }
        else if(root->val<key)
        {
            root->right=deleteNode(root->right,key);
        }
        else
        {
            if(root->right==NULL && root->left==NULL)
            {
                root=NULL;
                return root;

            }
            if(root->left!=NULL && root->right==NULL)
            {
                root=root->left;
                return root;
            }
            if(root->right!=NULL && root->left==NULL)
            {
                root=root->right;
                return root;
            }
            TreeNode *temp=lefty(root->right);
            int tempo=root->val;
            root->val=temp->val;
            temp->val=tempo;

            root->right=deleteNode(root->right,key);

            return root;
        }
        return root;
    }
};
```

https://leetcode.com/problems/delete-node-in-a-bst/



Inorder Successor in BST

Easy   Accuracy: **39.66%**   Submissions: **55987**   Points: **2**

Given a BST, and a reference to a Node x in the BST. Find the Inorder Successor of the given node in the BST.

Example 1:

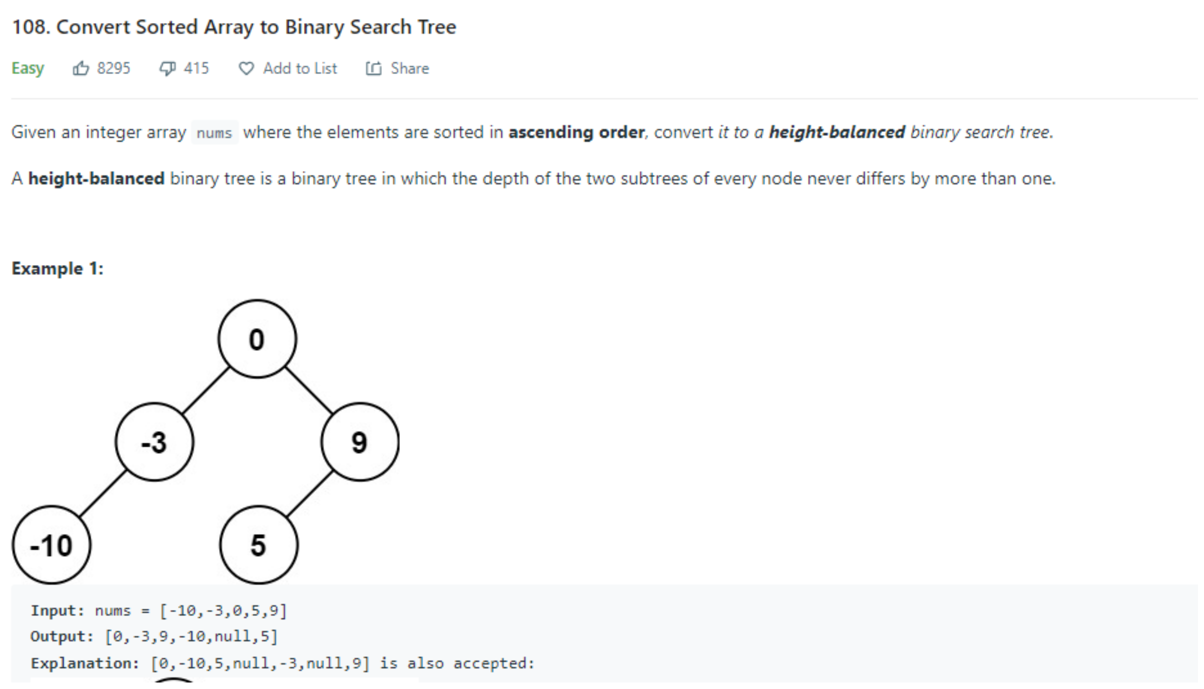
```
Input:
      2
    /   \
   1     3
K(data of x) = 2
Output: 3
Explanation:
Inorder traversal : 1 2 3
Hence, inorder successor of 2 is 3.
```

Example 2:

```
Input:
             20
            /   \
           8     22
          / \
         4   12
            /  \
           10   14
K(data of x) = 8
Output: 10
Explanation:
Inorder traversal: 4 8 10 12 14 20 22
```

```
class Solution{
  public:

    void solve(Node *root,vector<Node*> &in)
    {
        if(root==NULL)
        return;

        solve(root->left,in);
        in.push_back(root);
        solve(root->right,in);

    }
    // returns the inorder successor of the Node x in BST (rooted at 'root')
    Node * inOrderSuccessor(Node *root, Node *x)
    {
        //Your code here


        vector<Node*> in;
        solve(root,in);
        for(int i=0;i<in.size();i++)
        {
            if(in[i]==x && i<in.size()-1)
            return in[i+1];
        }
        return NULL;

    }
};
```

https://practice.geeksforgeeks.org/problems/inorder-successor-in-bst/1



108. Convert Sorted Array to Binary Search Tree

Easy 8295 415 Add to List Share

Given an integer array `nums` where the elements are sorted in **ascending order**, convert *it to a* ***height-balanced*** *binary search tree.*

A **height-balanced** binary tree is a binary tree in which the depth of the two subtrees of every node never differs by more than one.

Example 1:

```
Input: nums = [-10,-3,0,5,9]
Output: [0,-3,9,-10,null,5]
Explanation: [0,-10,5,null,-3,null,9] is also accepted:
```

```
class Solution {
public:

    TreeNode* binary(vector<int>& nums,int low,int high)
    {
        if(low>high)
        {
            return NULL;
        }

            int mid=(low+high)/2;
            TreeNode *temp=new TreeNode(nums[mid]);

            temp->left=binary(nums,low,mid-1);
            temp->right=binary(nums,mid+1,high);
        return temp;


    }
    TreeNode* sortedArrayToBST(vector<int>& nums) {

        int low=0;
        int high=nums.size()-1;


        if(nums.size()==0)
            return NULL;
        TreeNode* root=binary(nums,low,high);
        return root;


    }
};
```

https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree/