

Binary Search Tree problems

1.Recover Binary Search Tree

```
public class BinarySearchTree {  
    class Node  
    {  
        int key;  
        Node left,right;  
        public Node(int item)  
        {  
            key=item;  
            left=right=null;  
        }  
    }  
    Node root;  
    BinarySearchTree()  
    {  
        root=null;  
    }  
    BinarySearchTree(int value)  
    {  
        root=new Node(value);  
    }  
    void insert(int key)  
    {  
        root = insertRec(root, key);  
    }  
}
```

```

}
Node insertRec(Node root, int key)
{
    if (root == null)
    {
        root = new Node(key);
        return root;
    }
    else if (key < root.key)
        root.left = insertRec(root.left, key);
    else if (key > root.key)
        root.right = insertRec(root.right, key);
    return root;
}

void inorder()

{
    inorderRec(root);

}

void inorderRec(Node root)
{
    if (root != null)
    {
        inorderRec(root.left);
        System.out.println(root.key);
        inorderRec(root.right);
    }
}

```

```

    }
}

```

```
public static void main(String[] args) {
```

```
    BinarySearchTree tree = new BinarySearchTree();
```

```
    tree.insert(30);
```

```
    tree.insert(50);
```

```
    tree.insert(20);
```

```
    tree.insert(40);
```

```
    tree.insert(70);
```

```
    tree.inorder();
```

```

}

```

```

}

```

OUTPUT::

Success Details >

Runtime: 59 ms, faster than 5.25% of Java solutions for Recover Binary Search Tree.

Memory Usage: 48.4 MB, less than 5.46% of Java solutions for Recover Binary Search Tree.

Next challenges: [Binary Search Tree Iterator](#) [Web Crawler](#) [Count Nodes Equal to Sum of Descendants](#)

Show off your acceptance: [f](#) [t](#) [w](#) [in](#)

Time Submitted	Status	Runtime
09/05/2022 11:07	Accepted	59 ms

```

1 class Solution {
2     public void recoverTree(TreeNode root) {
3
4         //check whole tree until and unless it is valid
5         while(true){
6             boolean x = validate(root);
7             if(x) break;
8         }
9
10        //when we encounter two wrongly placed nodes , we swap them and return false
11        //we process this function until and unless it return true which means the tree is perfectly valid and there are no wrongly placed nodes left
12
13        Stack<TreeNode> stack = new Stack<TreeNode> ();
14        TreeNode cur = root;
15        while (stack.isEmpty() || cur != null) {
16            if (cur != null) {
17                stack.push(cur);
18                cur = cur.left;
19            }
20        }
21        //swap the two nodes
22        if (stack.size() > 1) {
23            TreeNode pre = stack.pop();
24            TreeNode next = stack.pop();
25            pre.val = next.val;
26            next.val = pre.val;
27        }
28    }
29
30    boolean validate(TreeNode root) {
31        if (root == null) return true;
32        if (root.left != null && root.left.val > root.val) return false;
33        if (root.right != null && root.right.val < root.val) return false;
34        return validate(root.left) && validate(root.right);
35    }
36}

```

Your previous code was restored from your local storage. [Reset to default](#)

Testcase	Run Code Result	Debugger
Accepted	Runtime: 0 ms	

Your input: [1, 3, null, null, 2]

Output: [3, 1, null, null, 2] [Diff](#)

Expected: [3, 1, null, null, 2]

Console [Use Example Testcases](#) [Run Code](#) [Submit](#)

2 . All Elements in Two Binary Search Trees

```
class Solution {  
    public List < Integer > getAllElements(TreeNode root1, TreeNode root2) {  
        Stack < TreeNode > s1 = new Stack();  
        while (root1 != null) {  
            s1.push(root1);  
            root1 = root1.left;  
        }  
  
        Stack < TreeNode > s2 = new Stack();  
        while (root2 != null) {  
            s2.push(root2);  
            root2 = root2.left;  
        }  
  
        List < Integer > result = new ArrayList();  
  
        find(result, s1, s2);  
        return result;  
    }  
    public void popStack(List < Integer > result, Stack < TreeNode > s) {  
        TreeNode n = s.pop();  
        result.add(n.val);  
        n = n.right;  
        while (n != null) {  
            s.push(n);  
            n = n.left;  
        }  
    }  
}
```

```

}

public void find(List < Integer > result, Stack < TreeNode > s1, Stack < TreeNode > s2) {

    while (!s1.isEmpty() || !s2.isEmpty()) {
        if (!s1.isEmpty() && !s2.isEmpty()) {
            if (s1.peek().val <= s2.peek().val) {
                popStack(result, s1);
            } else {
                popStack(result, s2);
            }
        } else if (!s1.isEmpty()) {
            while (!s1.isEmpty()) {
                popStack(result, s1);
            }
            break;
        } else {
            while (!s2.isEmpty()) {
                popStack(result, s2);
            }
            break;
        }
    }
}
}

```

OUTPUT::

LeetCode Explore Problems Interview **Contest** Discuss Store Premium 8

Desc... Solu... Discu... Sub...

Success Details >

Runtime: **40 ms**, faster than 32.75% of Java Elements in Two Binary Search Trees.

Memory Usage: **44.2 MB**, less than 94.64% All Elements in Two Binary Search Trees.

Next challenges:

Serialize and Deserialize Binary Tree **Cou...**

Validate Binary Tree Nodes

Show off your acceptance: f t in

Time Submitted	Status	Runtime
09/05/2022 11:11	Accepted	40 ms

Testcase Run Code Result Debugger

Accepted Runtime: 0 ms

Your input

```
[2,1,4]
[1,0,3]
```

Output

```
[0,1,1,2,3,4]
```

Expected

```
[0,1,1,2,3,4]
```

Console - Use Example Testcases Run Code Submit

```

11 s2.push(root2);
12 root2 = root2.left;
13 }
14
15 List < Integer > result = new ArrayList();
16
17 find(result, s1, s2);
18 return result;
19 }
20
21 public void popStack(List < Integer > result, Stack < TreeNode > s) {
22     TreeNode n = s.pop();
23     result.add(n.val);
24     n = n.right;
25     while (n != null) {
26         s.push(n);
27         n = n.left;
28     }
29 }
30
31 public void find(List < Integer > result, Stack < TreeNode > s1, Stack < TreeNode > s2) {
32     while (!s1.isEmpty() || !s2.isEmpty()) {
33         if (!s1.isEmpty() && !s2.isEmpty()) {
34             if (s1.peek().val <= s2.peek().val) {
35                 popStack(result, s1);
36             } else {

```

3.Find Leftmost and Rightmost nodes for a given node:

```

package skill6;

public class GFG {
    static void LeftRightNode(int preorder[], int n)
    {
        int min = Integer.MAX_VALUE, max = Integer.MIN_VALUE;

        for (int i = 0; i < n; i++)
        {
            if (min > preorder[i])
                min = preorder[i];

            if (max < preorder[i])
                max = preorder[i];
        }
        System.out.println("Leftmost node is " + min);
        System.out.println("Rightmost node is " + max);
    }

    public static void main(String[] args) {
        int preorder[] = { 3, 2, 1, 5, 4 };
        int n = 5;
        LeftRightNode(preorder, n);
    }
}

```

OUTPUT:

```
Window Help
GFG.java x
1 package skill6;
2
3 public class GFG {
4     static void LeftRightNode(int preorder[], int n)
5     {
6         int min = Integer.MAX_VALUE, max = Integer.MIN_VALUE;
7
8         for (int i = 0; i < n; i++)
9         {
10             if (min > preorder[i])
11                 min = preorder[i];
12
13             if (max < preorder[i])
14                 max = preorder[i];
15         }
16         System.out.println("Leftmost node is " + min);
17         System.out.println("Rightmost node is " + max);
18     }
19
20
21     public static void main(String[] args) {
22         int preorder[] = { 3, 2, 1, 5, 4 };
23         int n = 5;
24         LeftRightNode(preorder, n);
25
26     }
27
28
Console x
<terminated> GFG [Java Application] C:\Users\DELL\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe (05-Sep-2022, 11:19:26 am - 11:19:26 am)
Leftmost node is 1
Rightmost node is 5
```

4. Convert BST into Skewed Tree:

```
import java.io.*;
```

```
class Node
```

```
{
```

```
    int val;
```

```
    Node left, right;
```

```
    Node(int item)
```

```
{
```

```
        val = item;
```

```
        left = right = null;
```

```

    }
}

class GFG
{
    public static Node node;

    static Node prevNode = null;

    static Node headNode = null;

    static void flattenBTToSkewed(Node root,
                                   int order)
    {

        if(root == null)
        {
            return;
        }

        if(order > 0)
        {
            flattenBTToSkewed(root.right, order);
        }

        else
        {
            flattenBTToSkewed(root.left, order);
        }
    }
}

```



```
Node rightNode = root.right;
```

```
Node leftNode = root.left;
```

```
if(headNode == null)
```

```
{
```

```
    headNode = root;
```

```
    root.left = null;
```

```
    prevNode = root;
```

```
}
```

```
else
```

```
{
```

```
    prevNode.right = root;
```

```
    root.left = null;
```

```
    prevNode = root;
```

```
}
```

```
if (order > 0)
```

```
{
```

```
    flattenBTToSkewed(leftNode, order);
```

```
}
```

```
else
```

```
{
```

```
    flattenBTToSkewed(rightNode, order);
```

```
}
```

```
}
```

```

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

    System.out.print(root.val + " ");

    traverseRightSkewed(root.right);
}

public static void main (String[] args)
{

    GFG tree = new GFG();

    tree.node = new Node(5);

    tree.node.left = new Node(3);

    tree.node.right = new Node(6);

    int order = 0;

    flattenBTToSkewed(node, order);

    traverseRightSkewed(headNode);

}
}

```

OUTPUT::

main.cpp

Run

Output

```
1 import java.io.*;
2 class Node
3 {
4     int val;
5     Node left, right;
6     Node(int item)
7     {
8         val = item;
9         left = right = null;
10    }
11 }
12 class GFG
13 {
14     public static Node node;
15     static Node prevNode = null;
16     static Node headNode = null;
17     static void flattenBTTToSkewed(Node root,
18                                     int order)
19     {
20         if(root == null)
21         {
22             return;
23         }
24         if(order > 0)
25         {
26             flattenBTTToSkewed(root.right, order);
27         }
```

/tmp/eAtRA0Gazk.o

15 12 11 10 6 5 1