Sum Of Nodes

Send Feedback

For a given Binary Tree of integers, find and return the sum of all the nodes data.

Example:

```
                 10
               /      \
             20      30
           /    \
         40    50
```

When we sum up all the nodes data together, [10, 20, 30, 40 50] we get 150. Hence, the output will be 150.

Input Format:

The first and the only line of input will contain the nodes data, all separated by a single space. Since -1 is used as an indication whether the left or right node data exist for root, it will not be a part of the node data.

Output Format:

The first and the only line of output prints the sum of all the nodes data present in the binary tree.

Note:

You are not required to print anything explicitly. It has already been taken care of.

Constraints:

1 <= N <= 10^6
Where N is the total number of nodes in the binary tree.

Time Limit: 1 sec

Sample Input 1:

```
2 3 4 6 -1 -1 -1 -1 -1
```

Sample Output 1:

```
15
```

Sample Input 2:

```
1 2 3 4 5 6 7 -1 -1 -1 -1 -1 -1 -1 -1
```

Sample Output 2:

```
28
```

```
public class Solution {

	public static int getSum(BinaryTreeNode<Integer> root) {

		//Your code goes here.

        if(root==null){

            return 0;

        }

        int ans = getSum(root.left)+getSum(root.right);

        return root.data+ans;

	}

}
```

Preorder Binary Tree

Send Feedback

You are given the root node of a binary tree.Print its preorder traversal.

Input Format:

The first and the only line of input will contain the node data, all separated by a single space. Since -1 is used as an indication whether the left or right node data exist for root, it will not be a part of the node data.

Output Format:

The only line of output prints the preorder traversal of the given binary tree.

Constraints:

1 <= N <= 10^6
Where N is the total number of nodes in the binary tree.

Time Limit: 1 sec

Sample Input 1:

1 2 3 4 5 6 7 -1 -1 -1 -1 -1 -1 -1 -1

Sample Output 1:

1 2 4 5 3 6 7

Sample Input 2:

5 6 10 2 3 -1 -1 -1 -1 -1 9 -1 -1

Sample Output 1:

5 6 2 3 9 10

```
public class Solution {

        public static void preOrder(BinaryTreeNode<Integer> root) {

                //Your code goes here

    if(root==null){

      return;

    }

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

    preOrder(root.left);

    preOrder(root.right);

        }

}
```

Postorder Binary Tree

Send Feedback

For a given Binary Tree of integers, print the post-order traversal.

Input Format:

The first and the only line of input will contain the node data, all separated by a single space. Since -1 is used as an indication whether the left or right node data exist for root, it will not be a part of the node data.

Output Format:

The only line of output prints the post-order traversal of the given binary tree.

Constraints:

1 <= N <= 10^6
Where N is the total number of nodes in the binary tree.

Time Limit: 1 sec

Sample Input 1:

1 2 3 4 5 6 7 -1 -1 -1 -1 -1 -1 -1 -1

Sample Output 1:

4 5 2 6 7 3 1

Sample Input 2:

```
5 6 10 2 3 -1 -1 -1 -1 -1 9 -1 -1
```

Sample Output 1:

```
2 9 3 6 10 5
```

```
public class Solution {

	public static void postOrder(BinaryTreeNode<Integer> root) {

		//Your code goes here

      if(root==null){

         return;

      }

      postOrder(root.left);

      postOrder(root.right);

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

	}

}
```

Nodes Greater Than X

Send Feedback

For a given a binary tree of integers and an integer X, find and return the total number of nodes of the given binary tree which are having data greater than X.

Input Format:

The first line of input will contain the node data, all separated by a single space. Since -1 is used as an indication whether the left or right node data exist for root, it will not be a part of the node data.

The second line of input contains an integer, denoting the value of X.

Note:

You are not required to print anything explicitly. It has already been taken care of.

Output Format:

The only line of output prints the total number of nodes where the node data is greater than X.

Constraints:

1 <= N <= 10^5
Where N is the total number of nodes in the binary tree.

Time Limit: 1 sec

Sample Input 1:

```
1 4 2 3 -1 -1 -1
2
```

Sample Output 1:

```
2
```

Explanation for Sample Input 1:

Out of the four nodes of the given binary tree, [3, 4] are the node data that are greater than X = 2.

Sample Input 2:

```
5 6 10 2 3 -1 -1 -1 -1 -1 9 -1 -1
5
```

Sample Output 2:

```
3
```

```
blic class Solution {
        public static int countNodesGreaterThanX(BinaryTreeNode<Integer> root, int x) {

                //Your code goes here

    if(root==null){

        return 0;

    }

    int ans =countNodesGreaterThanX(root.left,x)+countNodesGreaterThanX(root.right,x);

    if(root.data>x){

        return ans+1;

    }

    else{

        return ans;

    }

        }



}
```

Height Of Tree

Send Feedback

For a given Binary Tree of integers, find and return the height of the tree.

Example:

```
                        10
                      /      \
                    20      30
                  /    \
                40    50
```

Height of the given tree is 3.

Height is defined as the total number of nodes along the longest path from the root to any of the leaf node.

Input Format:

The first and the only line of input will contain the node data, all separated by a single space. Since -1 is used as an indication whether the left or right node data exist for root, it will not be a part of the node data.

Output Format:

The first and the only line of output prints the height of the given binary tree.

Note:

You are not required to print anything explicitly. It has already been taken care of.

Constraints:

0 <= N <= 10^5

Where N is the total number of nodes in the binary tree.

Time Limit: 1 sec

Sample Input 1:

```
10 20 30 40 50 -1 -1 -1 -1 -1 -1
```

Sample Output 1:

```
3
```

Sample Input 2:

```
3 -1 -1
```

Sample Output 2:

```
1
```

```
public class Solution {



        public static int height(BinaryTreeNode<Integer> root) {

                //Your code goes here

    if(root==null){

      return 0;

    }

    int leftOutput = height(root.left);

    int rightOutput = height(root.right);

    if(leftOutput>rightOutput){

      return leftOutput+1;

    }

    else{

      return rightOutput+1;

    }

        }



}
```

Replace Node With Depth

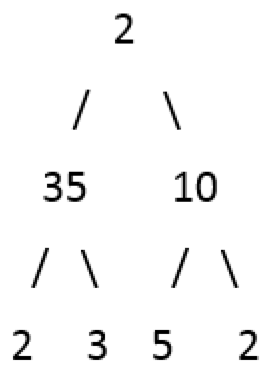
Send Feedback

For a given a Binary Tree of integers, replace each of its data with the depth of the tree.

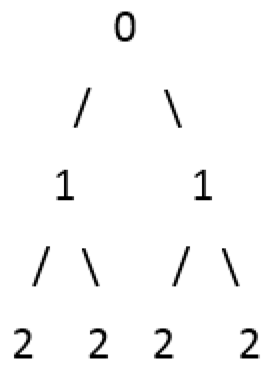
Root is at depth 0, hence the root data is updated with 0. Replicate the same further going down the in the depth of the given tree.

The modified tree will be printed in the in-order fashion.

Example:



The above tree after updating will look like this:



Output: 2 1 2 0 2 1 2 (printed in the in-order fashion)

Input Format:

The first and the only line of input will contain the node data, all separated by a single space. Since -1 is used as an indication whether the left or right node data exist for root, it will not be a part of the node data.

Output Format:

The first and the only line of output prints the updated tree in the in-order fashion.

Note:

You are not required to print anything explicitly. It has already been taken care of.

Constraints:

1 <= N <= 10^5
Where N is the total number of nodes in the binary tree.

Time Limit: 1sec

Sample Input 1:

```
2 35 10 2 3 5 2 -1 -1 -1 -1 -1 -1 -1 -1
```

Sample Output 1:

```
2 1 2 0 2 1 2
```

Sample Input 2:

```
1 2 3 4 5 6 7 -1 -1 -1 -1 -1 -1 -1 -1
```

Sample Output 2:

```
2 1 2 0 2 1 2
```

```
public class Solution {



	public static void changeToDepthTree(BinaryTreeNode<Integer> root) {

	    //Your code goes here

     changeToDepthTree(root,0);

	}

    public static void changeToDepthTree(BinaryTreeNode<Integer> root, int depth){

        if(root==null){

            return;

        }

        root.data = depth;

        changeToDepthTree(root.left,depth+1);

        changeToDepthTree(root.right,depth+1);

    }



}
```

Is Node Present?

Send Feedback

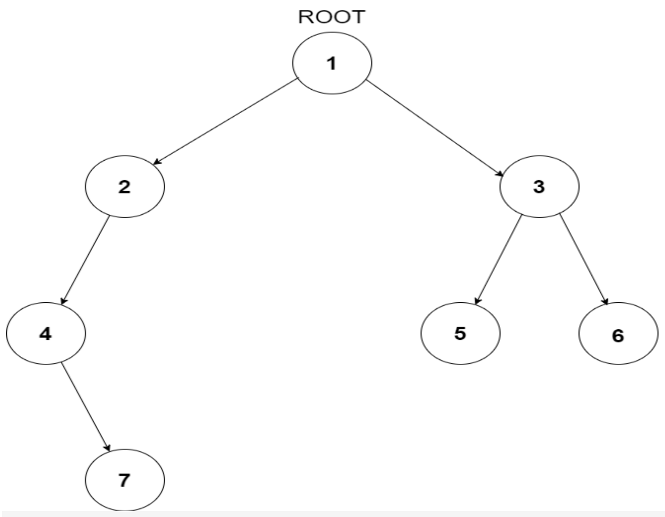
For a given Binary Tree of type integer and a number X, find whether a node exists in the tree with data X or not.

Input Format:

The first line of each test case contains elements of the first tree in the level order form. The line consists of values of nodes separated by a single space. In case a node is null, we take -1 in its place.

The second line of each test case contains the node value you have to find.

For example, the input for the tree depicted in the below image would be:



```
1
2 3
4 -1 5 6
-1 7 -1 -1 -1 -1
-1 -1
```

Explanation:

Level 1:
The root node of the tree is 1

Level 2:
Left child of 1 = 2
Right child of 1 = 3

Level 3:
Left child of 2 = 4
Right child of 2 = null (-1)
Left child of 3 = 5
Right child of 3 = 6

Level 4:
Left child of 4 = null (-1)
Right child of 4 = 7
Left child of 5 = null (-1)
Right child of 5 = null (-1)
Left child of 6 = null (-1)
Right child of 6 = null (-1)

Level 5:
Left child of 7 = null (-1)
Right child of 7 = null (-1)

The first not-null node(of the previous level) is treated as the parent of the first two nodes of the current level. The second not-null node (of the previous level) is treated as the parent node for the next two nodes of the current level and so on.
The input ends when all nodes at the last level are null(-1).

Note:

The above format was just to provide clarity on how the input is formed for a given tree.
The sequence will be put together in a single line separated by a single space. Hence, for the above-depicted tree, the input will be given as:

```
1 2 3 4 -1 5 6 -1 7 -1 -1 -1 -1 -1 -1
```

Output Format:

The only line of output prints 'true' or 'false'.

The output of each test case should be printed in a separate line.

Note:

You are not required to print anything explicitly. It has already been taken care of.

Constraints:

```
1 <= N <= 10^5

Where N is the total number of nodes in the binary tree.

Time Limit: 1 sec.
```

Sample Input 1:

```
8 3 10 1 6 -1 14 -1 -1 4 7 13 -1 -1 -1 -1 -1 -1 -1
7
```

Sample Output 1:

```
true
```

Explanation For Output 1:

Clearly, we can see that 7 is present in the tree. So, the output will be true.

Sample Input 2:

```
2 3 4 -1 -1 -1 -1
10
```

Sample Output 2:

```
false
```

```
public class Solution {



        public static boolean isNodePresent(BinaryTreeNode<Integer> root, int x) {

            //Your code goes here

    if(root==null){

        return false;

    }

    if(root.data==x){

        return true;

    }

    else{



        return (isNodePresent(root.left,x)||isNodePresent(root.right,x));

    }

        }



}
```

Nodes without sibling

Send Feedback

For a given Binary Tree of type integer, print all the nodes without any siblings.

Input Format:

The first and the only line of input will contain the node data, all separated by a single space. Since -1 is used as an indication whether the left or right node data exist for root, it will not be a part of the node data.

Output Format:

The only line of output prints the node data in a top to down fashion with reference to the root.
Node data in the left subtree will be printed first and then the right subtree.
A single space will separate them all.

Constraints:

1 <= N <= 10^5
Where N is the total number of nodes in the binary tree.

Time Limit: 1 second

Sample Input 1:

```
5 6 10 2 3 -1 -1 -1 -1 -1 9 -1 -1
```

Sample Output 1:

```
9
```

Sample Input 2:

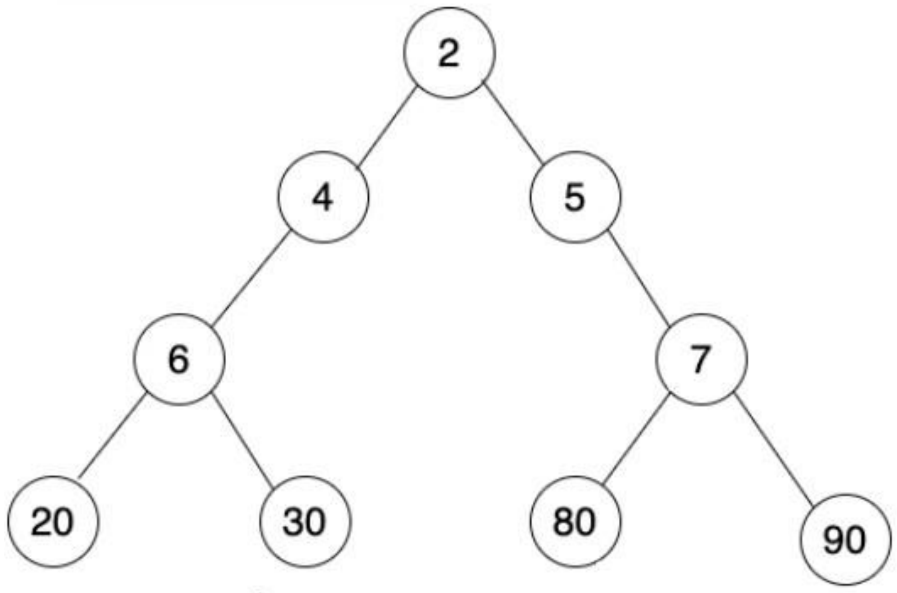
```
2 4 5 6 -1 -1 7 20 30 80 90 -1 -1 -1 -1 -1 -1 -1 -1
```

Sample Output 2:

```
6 7
```

Explanation of Sample Input 2:

The input tree when represented in a two-dimensional plane, it would look like this:



In respect to the root, node data in the left subtree that satisfy the condition of not having a sibling would be 6, taken in a top-down sequence. Similarly, for the right subtree, 7 is the node data without any sibling.

Since we print the siblings in the left-subtree first and then the siblings from the right subtree, taken in a top-down fashion, we print 6 7.

```
import java.util.*;

public class Solution {



        public static void printNodesWithoutSibling(BinaryTreeNode<Integer> root) {

                //Your code goes here

      if (root==null)
```

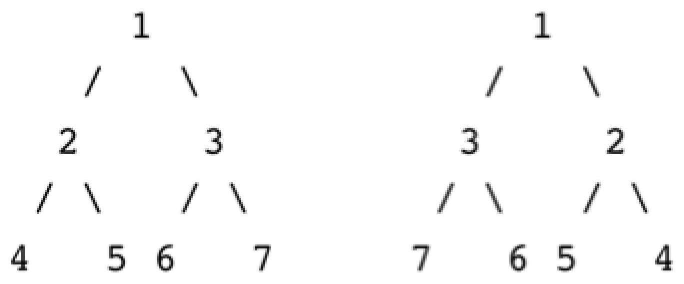
```
        {
           return;
        }

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

        if (root.left==null)
        {
           System.out.print(root.right.data+" ");
           printNodesWithoutSibling(root.right);
        }
        else if (root.right==null)
        {
           System.out.print(root.left.data+" ");
           printNodesWithoutSibling(root.left);

        }
        else
        {
           printNodesWithoutSibling(root.left);
           printNodesWithoutSibling(root.right);
        }

            }

}
```

Mirror Binary Tree

Send Feedback

For a given Binary Tree of type integer, update it with its corresponding mirror image.

Example:



Original BT   Mirrored BT

Input Format:

The first and the only line of input will contain the node data, all separated by a single space. Since -1 is used as an indication whether the left or right node data exist for root, it will not be a part of the node data.

Output Format:

The only line of output prints the mirrored tree in a level-wise order.
Each level will be printed on a new line. Elements printed at each level will be separated by a single line.

Note:

You are not required to print anything explicitly. It has already been taken care of.

Constraints:

1 <= N <= 10^5
Where N is the total number of nodes in the binary tree.

Time Limit: 1 sec

Sample Input 1:

```
1 2 3 4 5 6 7 -1 -1 -1 -1 -1 -1 -1 -1
```

Sample Output 1:

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

Sample Input 2:

```
5 10 6 2 3 -1 -1 -1 -1 -1 9 -1 -1
```

Sample Output 2:

```
5
6 10
3 2
9
```

```
public class Solution {


        public static void mirrorBinaryTree(BinaryTreeNode<Integer> root){

                //Your code goes here
        if(root==null)

           return;

        BinaryTreeNode<Integer> temp = root.right;

        root.right =root.left;

        root.left = temp;


        mirrorBinaryTree(root.left);

        mirrorBinaryTree(root.right);




            }


}
```

Lecture 11: Binary Trees/is Balanced

```
// is
Balanced
           // Send Feedback
           // Given a binary tree, check if its balanced i.e. depth of left and right subtrees
           of every node differ by at max 1. Return true if given binary tree is balanced,
           false otherwise.
           // Input format :
           // Elements in level order form (separated by space). If any node does not have
           left or right child, take -1 in its place.
           // Sample Input 1 :
           // 5 6 10 2 3 -1 -1 -1 -1 -1 9 -1 -1
           // Sample Output 1 :
           // false
           // Sample Input 2 :
           // 1 2 3 -1 -1 -1 -1
           // Sample Output 2 :
           // true
           import java.util.*;
           public class Solution {
           /*      Binary Tree Node class
            *
            * class BinaryTreeNode<T> {
                          T data;
                          BinaryTreeNode<T> left;
                      BinaryTreeNode<T> right;

                      public BinaryTreeNode(T data) {
                                 this.data = data;
                      }
                  }
                  */

              public static  int height(BinaryTreeNode<Integer> root)
               {
                   /* base case tree is empty */
                   if (root == null)
                       return 0;

                   /* If tree is not empty then height = 1 + max of left
                    height and right heights */
                   return 1 + Math.max(height(root.left), height(root.right));
               }



                  public static boolean checkBalanced(BinaryTreeNode<Integer> root){


                    int lh; /* for height of left subtree */

                   int rh; /* for height of right subtree */

                   /* If tree is empty then return true */
                   if (root == null)
                       return true;

                   /* Get the height of left and right sub trees */
                   lh = height(root.left);
                   rh = height(root.right);

                   if (Math.abs(lh - rh) <= 1
                       && checkBalanced(root.left)
                       && checkBalanced(root.right))
                       return true;

                   /* If we reach here then tree is not height-balanced */
                   return false;




                  }
```

```
}
```

Diameter Of Binary Tree

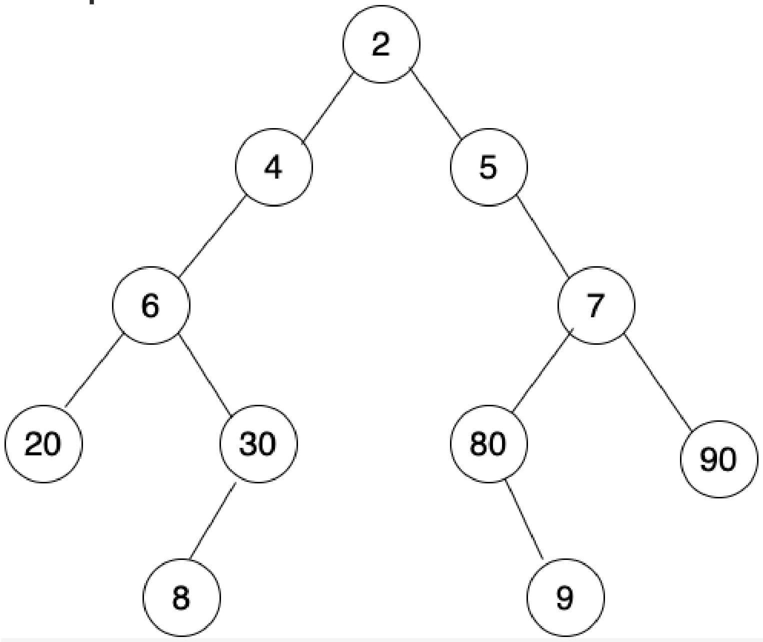
Send Feedback

For a given Binary of type integer, find and return the 'Diameter'.

Diameter of a Tree

The diameter of a tree can be defined as the maximum distance between two leaf nodes. Here, the distance is measured in terms of the total number of nodes present along the path of the two leaf nodes, including both the leaves.

Example:



The maximum distance can be seen between the leaf nodes 8 and 9.
The distance is 9 as there are a total of nine nodes along the longest path from 8 to 9(inclusive of both). Hence the diameter according to the definition will be 9.

Input Format:

The first and the only line of input will contain the node data, all separated by a single space. Since -1 is used as an indication whether the left or right node data exist for root, it will not be a part of the node data.

Output Format:

The only line of output prints an integer, representing the diameter of the tree.

Note:

You are not required to print anything explicitly. It has already been taken care of.

Constraints:

1 <= N <= 10^5
Where N is the total number of nodes in the binary tree.

Time Limit: 1 sec

Sample Input 1:

```
2 4 5 6 -1 -1 7 20 30 80 90 -1 -1 8 -1 -1 9 -1 -1 -1 -1 -1 -1
```

Sample Output 1:

```
9
```

Sample Input 2:

```
1 2 3 4 5 6 7 -1 -1 -1 -1 -1 -1 -1 -1
```

Sample Output 2:

```
5
```

```
public class Solution {
        public static int diameterOfBinaryTree(BinaryTreeNode<Integer> root){
               //Your code goes here
         if(root==null){
             return 0;
         }
         return findHeight(root.left)+findHeight(root.right)+1;
        }
    public static int findHeight(BinaryTreeNode<Integer> root){
        if(root==null){
            return 0;
        }
        int leftHeight = findHeight(root.left);
        int rightHeight= findHeight(root.right);

        if(leftHeight>rightHeight){
            return leftHeight+1;
        }
        else{
            return rightHeight+1;
        }
    }


}
```

Construct Tree Using Inorder and Preorder

Send Feedback

For a given preorder and inorder traversal of a Binary Tree of type integer stored in an array/list, create the binary tree using the given two arrays/lists. You just need to construct the tree and return the root.

Note:

Assume that the Binary Tree contains only unique elements.

Input Format:

The first line of input contains an integer N denoting the size of the list/array. It can also be said that N is the total number of nodes the binary tree would have.

The second line of input contains N integers, all separated by a single space. It represents the preorder-traversal of the binary tree.

The third line of input contains N integers, all separated by a single space. It represents the inorder-traversal of the binary tree.

Output Format:

The given input tree will be printed in a level order fashion where each level will be printed on a new line.
Elements on every level will be printed in a linear fashion. A single space will separate them.

Constraints:

1 <= N <= 10^4
Where N is the total number of nodes in the binary tree.

Time Limit: 1 sec

Sample Input 1:

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

Sample Output 1:

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

Sample Input 2:

```
6
5 6 2 3 9 10
2 6 3 9 5 10
```

Sample Output 2:

```
5
6 10
2 3
9
```

```
public class Solution {


	public static BinaryTreeNode<Integer> buildTree(int[] preOrder, int[]
inOrder) {
		//Your code goes here
	 BinaryTreeNode<Integer> root = buildTree(preOrder, inOrder, 0
,preOrder.length-1, 0, inOrder.length-1);
	return root;


	}


    public static BinaryTreeNode<Integer> buildTree(int[] preorder, int[]
inorder,int siPre, int eiPre, int siIn, int eiIn)
    {
	 //Base case - If number of elements in the pre-order is 0
	if (siPre>eiPre)
	{
		return null;
	}

	//Defining the root node for current recursion
	int rootData=preorder[siPre];
	BinaryTreeNode<Integer> root = new BinaryTreeNode<Integer>(rootData);

	//Finding root data's location in Inorder (Assuming root data exists in
Inorder)
	int rootIndexInorder=-1;
	for (int i=siIn;i<=eiIn;i++)
	{
		if (rootData==inorder[i])
		{
			rootIndexInorder=i;
			break;
		}
	}

	//Defining index limits for Left Subtree Inorder
	int siInLeft=siIn;
```

```
        int eiInLeft=rootIndexInorder-1;

        //Defining the index limits for Left Subtree Preorder
        int siPreLeft=siPre+1;
        int leftSubTreeLength = eiInLeft - siInLeft + 1;
        int eiPreLeft=(siPreLeft)+(leftSubTreeLength-1);

        //Defining index limits for Right Subtree Inorder
        int siInRight=rootIndexInorder+1;
        int eiInRight=eiIn;

        //Defining index limits for Right Subtree Preorder
        int siPreRight=eiPreLeft+1;
        int eiPreRight=eiPre;

        BinaryTreeNode<Integer> leftChild = buildTree(preorder, inorder, siPreLeft,
eiPreLeft, siInLeft, eiInLeft);
        BinaryTreeNode<Integer> rightChild = buildTree(preorder, inorder,
siPreRight, eiPreRight, siInRight, eiInRight);
        root.left=leftChild;
        root.right=rightChild;
        return root;
        }

}
```

Construct Tree Using Inorder and PostOrder

Send Feedback

For a given postorder and inorder traversal of a Binary Tree of type integer stored in an array/list, create the binary tree using the given two arrays/lists. You just need to construct the tree and return the root.

Note:

Assume that the Binary Tree contains only unique elements.

Input Format:

The first line of input contains an integer N denoting the size of the list/array. It can also be said that N is the total number of nodes the binary tree would have.

The second line of input contains N integers, all separated by a single space. It represents the Postorder-traversal of the binary tree.

The third line of input contains N integers, all separated by a single space. It represents the inorder-traversal of the binary tree.

Output Format:

The given input tree will be printed in a level order fashion where each level will be printed on a new line.
Elements on every level will be printed in a linear fashion. A single space will separate them.

Constraints:

1 <= N <= 10^4
Where N is the total number of nodes in the binary tree.

Time Limit: 1 sec

Sample Input 1:

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

Sample Output 1:

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

Sample Input 2:

```
6
2 9 3 6 10 5
2 6 3 9 5 10
```

Sample Output 2:

```
5
6 10
2 3
9
```

```
public class Solution {


       public static BinaryTreeNode<Integer> buildTree(int[] postOrder, int[]
inOrder) {
              //Your code goes here
        BinaryTreeNode<Integer> root = buildTree(postOrder, inOrder, 0
,postOrder.length-1, 0, inOrder.length-1);
       return root;
       }

    public static BinaryTreeNode<Integer> buildTree(int[] postOrder, int[]
inOrder,int siPost, int eiPost, int siIn, int eiIn) {
              // TODO Auto-generated method stub

       //Base case - If number of elements in the post-order is 0
       if (siPost>eiPost)
       {
              return null;
       }

       //Defining the root node for current recursion
       int rootData=postOrder[eiPost];
       BinaryTreeNode<Integer> root = new BinaryTreeNode<Integer>(rootData);

       //Finding root data's location in Inorder (Assuming root data exists in
Inorder)
       int rootIndex=-1;
       for (int i=siIn;i<=eiIn;i++)
       {
              if (rootData==inOrder[i])
              {
               rootIndex=i;
                     break;
              }
       }
```

```
        //Defining index limits for Left Subtree Inorder
        int siInLeft=siIn;
        int eiInLeft=rootIndex-1;

        //Defining the index limits for Left Subtree Preorder
        int siPostLeft=siPost;
        int leftSubTreeLength = eiInLeft - siInLeft + 1;
        int eiPostLeft=(siPostLeft)+(leftSubTreeLength-1);

        //Defining index limits for Right Subtree Inorder
        int siInRight=rootIndex+1;
        int eiInRight=eiIn;

        //Defining index limits for Right Subtree Preorder
        int siPostRight=eiPostLeft+1;
        int eiPostRight=eiPost-1;

         BinaryTreeNode<Integer> rightChild = buildTree(postOrder, inOrder,
siPostRight, eiPostRight, siInRight, eiInRight);
        BinaryTreeNode<Integer> leftChild = buildTree(postOrder, inOrder,
siPostLeft, eiPostLeft, siInLeft, eiInLeft);

        root.left=leftChild;
        root.right=rightChild;
        return root;
        }

}
```

INORDER PREORDER

```
class BinaryTreeNode:
    def __init__(self,data):
        self.data=data;
        self.left=None
        self.right=None

def printTree(root):
    if root==None:
        return
    print(root.data)
    printTree(root.left)
    printTree(root.right)

def printTreeDetailed(root):
    if root==None:
        return
    print(root.data,end=":")
    if root.left!=None:
        print("L",root.left.data,end=",")
    if root.right!=None:
        print("R",root.right.data,end="")
    print()
    printTreeDetailed(root.left)
    printTreeDetailed(root.right)

def buildTreeFromPreIn(pre,inorder):
    if len(pre)==0:
        return None
    rootData=pre[0]
    root=BinaryTreeNode(rootData)
    rootIndexInInOrder=-1
    for i in range(0,len(inorder)):
        if inorder[i]==rootData:
            rootIndexInInorder=i
            break
    if rootIndexInInorder==-1:
        return None

    leftInorder=inorder[0:rootIndexInInorder]
    rightInorder=inorder[rootIndexInInorder+1:]

    lenLeftSubtree=len(leftInorder)

    leftPreorder=pre[1:lenLeftSubtree+1]
    rightPreorder=pre[lenLeftSubtree+1:]

    leftChild=buildTreeFromPreIn(leftPreorder,leftInorder)
    rightChild=buildTreeFromPreIn(rightPreorder,rightInorder)

    root.left=leftChild
    root.right=rightChild
    return root

pre=[1,2,4,5,3,6,7]
inorder=[4,2,5,1,6,3,7]
root=buildTreeFromPreIn(pre,inorder)
printTreeDetailed(root)
```

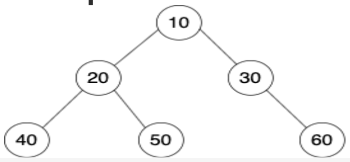
Create & Insert Duplicate Node

Send Feedback

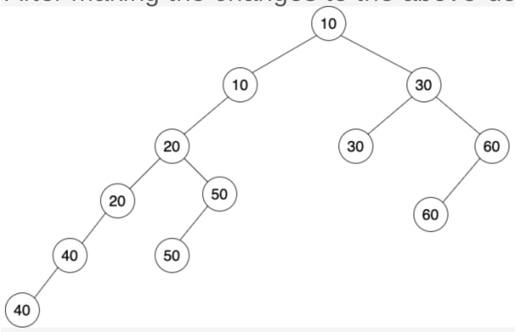
For a given a Binary Tree of type integer, duplicate every node of the tree and attach it to the left of itself.

The root will remain the same. So you just need to insert nodes in the given Binary Tree.

Example:



After making the changes to the above-depicted tree, the updated tree will look like this.



You can see that every node in the input tree has been duplicated and inserted to the left of itself.

Input format :

The first and the only line of input will contain the node data, all separated by a single space. Since -1 is used as an indication whether the left or right node data exist for root, it will not be a part of the node data.

Output Format :

The updated tree will be printed in a level order fashion where each level will be printed on a new line. Elements on every level will be printed in a linear fashion. A single space will separate them.

Note:

You are not required to print anything explicitly. It has already been taken care of. Just implement the function to achieve the desired structure of the tree.

Constraints :

1 <= N <= 10^5
Where N is the total number of nodes in the binary tree.

Time Limit: 1 sec

Sample Input 1:

```
10 20 30 40 50 -1 60 -1 -1 -1 -1 -1 -1
```

Sample Output 1:

```
10
10 30
20 30 60
20 50 60
40 50
40
```

Sample Input 2:

```
8 5 10 2 6 -1 -1 -1 -1 -1 7 -1 -1
```

Sample Output 2:

```
8
8 10
5 10
5 6
2 6 7
2 7
```

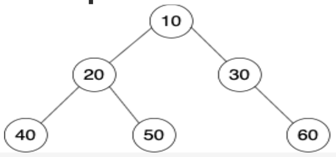
```
public class Solution {

	public static void insertDuplicateNode(BinaryTreeNode<Integer> root) {

		//Your code goes here

    if(root==null)

        return;

    BinaryTreeNode<Integer> duplicateNode = new BinaryTreeNode<Integer>(root.data);

    //duplicateNode.data=root.data;

    BinaryTreeNode<Integer> temp = root.left;

    root.left = duplicateNode;

    duplicateNode.left = temp;

    insertDuplicateNode(root.left.left);

    insertDuplicateNode(root.right);
	}


}
```

Level order traversal

Send Feedback

For a given a Binary Tree of type integer, print it in a level order fashion where each level will be printed on a new line. Elements on every level will be printed in a linear fashion and a single space will separate them.

Example:



For the above-depicted tree, when printed in a level order fashion, the output would look like:

```
10
20 30
40 50 60
```
Where each new line denotes the level in the tree.

Input Format:

The first and the only line of input will contain the node data, all separated by a single space. Since -1 is used as an indication whether the left or right node data exist for root, it will not be a part of the node data.

Output Format:

The given input tree will be printed in a level order fashion where each level will be printed on a new line.
Elements on every level will be printed in a linear fashion. A single space will separate them.

Constraints:

1 <= N <= 10^5
Where N is the total number of nodes in the binary tree.

Time Limit: 1 sec

Sample Input 1:

```
10 20 30 40 50 -1 60 -1 -1 -1 -1 -1 -1
```

Sample Output 1:

```
10
20 30
40 50 60
```

Sample Input 2:

```
8 3 10 1 6 -1 14 -1 -1 4 7 13 -1 -1 -1 -1 -1 -1 -1
```

Sample Output 2:

```
8
3 10
1 6 14
4 7 13
```

```
import java.util.*;

public class Solution {



	public static void printLevelWise(BinaryTreeNode<Integer> root) {

		//Your code goes here
```

```
if (root==null)
   return;


Queue<BinaryTreeNode<Integer>> nodesToPrint = new LinkedList<BinaryTreeNode<Integer>>();
nodesToPrint.add(root);
nodesToPrint.add(null);
while(!nodesToPrint.isEmpty())
{
   BinaryTreeNode<Integer> front=nodesToPrint.poll();
   if (front==null)
   {
      if (nodesToPrint.isEmpty())
         break;
      else
      {
         System.out.println();
                nodesToPrint.add(null);
      }


   }
   else
   {
      System.out.print(front.data+" ");
      if (front.left!=null)
         nodesToPrint.add(front.left);
      if (front.right!=null)
         nodesToPrint.add(front.right);
   }
}
```

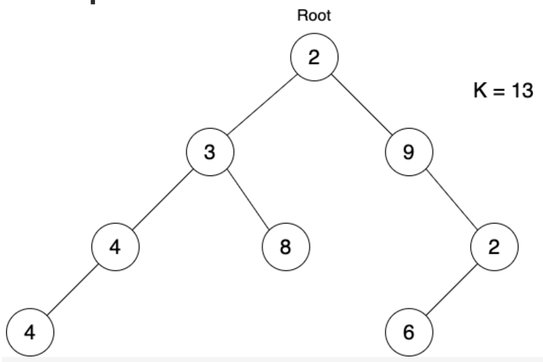
```
        }


}
```

Path Sum Root to Leaf

Send Feedback

For a given Binary Tree of type integer and a number K, print out all root-to-leaf paths where the sum of all the node data along the path is equal to K.

Example:



If you see in the above-depicted picture of Binary Tree, we see that there are a total of two paths, starting from the root and ending at the leaves which sum up to a value of K = 13.

The paths are:
a. 2 3 4 4
b. 2 3 8

One thing to note here is, there is another path in the right sub-tree in reference to the root, which sums up to 13 but since it doesn't end at the leaf, we discard it.
The path is: 2 9 2(not a leaf)

Input Format:

The first line of input will contain the node data, all separated by a single space. Since -1 is used as an indication whether the left or right node data exist for root, it will not be a part of the node data.

The second line of input contains an integer value K.

Output Format:

Lines equal to the total number of paths will be printed. All the node data in every path will be printed in a linear fashion taken in the order they appear from top to down bottom in the tree. A single space will separate them all.

Constriants:

1 <= N <= 10^5
0 <= K <= 10^8
Where N is the total number of nodes in the binary tree.

Time Limit: 1 second

Sample Input 1:

```
2 3 9 4 8 -1 2 4 -1 -1 -1 6 -1 -1 -1 -1 -1
13
```

Sample Output 1:

```
2 3 4 4
2 3 8
```

Sample Input 2:

```
5 6 7 2 3 -1 1 -1 -1 -1 9 -1 -1 -1 -1
13
```

Sample Output 2:

```
5 6 2
```

```
5 7 1
import java.util.*;

public class Solution {

            public static void rootToLeafPathsSumToK(BinaryTreeNode<Integer> root, int k) {

                        //Your code goes here

       String arr="";

      rootToLeafPathsSumToK(root,k,arr);

            }


   public static void rootToLeafPathsSumToK(BinaryTreeNode<Integer> root, int k,String arr)

   {

      if (root==null)

      {

         return;

      }


      int rootData=root.data;

      //System.out.println("Root data: "+rootData);

      //System.out.println("k: "+k);

      //System.out.println("Old Arraylist: "+arr);

      arr=arr+rootData+" ";

      if(k==rootData && root.left==null && root.right==null)

      {

         //System.out.print("Path found: ");

         //for (int i=0;i<arr.length();i++)

           //System.out.print(arr.charAt(i)+" ");

         //System.out.println();

         System.out.println(arr);

         return;

      }
```

```
        //System.out.println();



        rootToLeafPathsSumToK(root.left,k-rootData,arr);

        rootToLeafPathsSumToK(root.right,k-rootData,arr);



    }



}
```

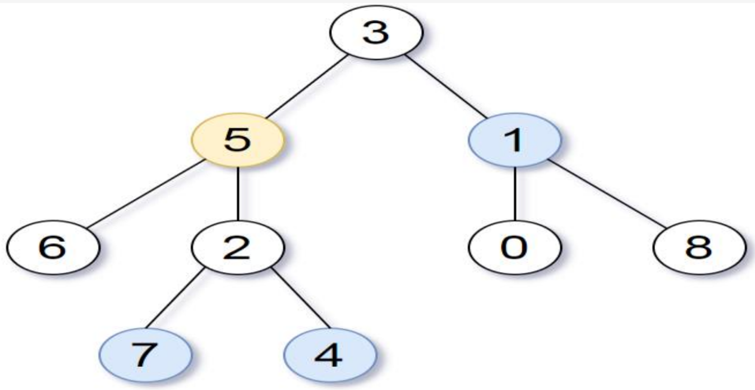
Print nodes at distance k from node

Send Feedback

You are given a Binary Tree of type integer, a target node, and an integer value K.
Print the data of all nodes that have a distance K from the target node. The order in which they would be printed will not matter.

Example:

For a given input tree(refer to the image below):
1. Target Node: 5
2. K = 2



Starting from the target node 5, the nodes at distance K are 7 4 and 1.

Input Format:

The first line of input will contain the node data, all separated by a single space. Since -1 is used as an indication whether the left or right node data exist for root, it will not be a part of the node data.

The second line of input contains two integers separated by a single space, representing the value of the target node and K, respectively.

Output Format:

All the node data at distance K from the target node will be printed on a new line.

The order in which the data is printed doesn't matter.

Constraints:

1 <= N <= 10^5
Where N is the total number of nodes in the binary tree.

Time Limit: 1 sec

Sample Input 1:

```
5 6 10 2 3 -1 -1 -1 -1 -1 9 -1 -1
3 1
```

Sample Output 1:

```
9
6
```

Sample Input 2:

```
1 2 3 4 5 6 7 -1 -1 -1 -1 -1 -1 -1 -1
3 3
```

Sample Output 2:

```
4
5
```

```
public class Solution {



        public static void nodesAtDistanceK(BinaryTreeNode<Integer> root, int node, int k) {

            //Your code goes here

    print(root,node,k);

        }

  private static int print(BinaryTreeNode<Integer> root,int node,int k){

    if(root==null)

      return -1;

    int rootData = root.data;

    if(rootData==node){

      printNodesAtDistanceK(root,k);

      return 0;

    }



    int leftSubTreeDist = 0,rightSubTreeDist = 0;



    leftSubTreeDist = print(root.left,node,k);



    if(leftSubTreeDist !=-1){

      if(leftSubTreeDist +1==k){

        System.out.println(rootData);

      }

      else{

        rightSubTreeDist = k-(leftSubTreeDist+1)-1;
```

```
            printNodesAtDistanceK(root.right, rightSubTreeDist);

        }

        return leftSubTreeDist+1;

    }

    rightSubTreeDist=print(root.right,node,k);

    if (rightSubTreeDist!=-1)

    {

        if(rightSubTreeDist+1==k)

        {

            System.out.println(rootData);

        }

        else

        {

            leftSubTreeDist=k-(rightSubTreeDist+1)-1;

            printNodesAtDistanceK(root.left, leftSubTreeDist);

        }

        return rightSubTreeDist+1;

    }

    return -1;

}


private static void printNodesAtDistanceK(BinaryTreeNode<Integer> root, int k)

{

    if (root==null || k<0)

          return;


    if (k == 0)

    {

        System.out.println(root.data);
```

```
            return;

        }



        printNodesAtDistanceK(root.left,k-1);

        printNodesAtDistanceK(root.right,k-1);



    }



}
```

Minimum and Maximum in the Binary Tree

Send Feedback

For a given a Binary Tree of type integer, find and return the minimum and the maximum data values.

Return the output as an object of Pair class, which is already created.

Note:

All the node data will be unique and hence there will always exist a minimum and maximum node data.

Input Format:

The first and the only line of input will contain the node data, all separated by a single space. Since -1 is used as an indication whether the left or right node data exist for root, it will not be a part of the node data.

Output Format:

The only line of output prints two integers denoting the minimum and the maximum data values respectively. A single line will separate them both.

Constraints:

2 <= N <= 10^5
Where N is the total number of nodes in the binary tree.

Time Limit: 1 sec

Sample Input 1:

```
8 3 10 1 6 -1 14 -1 -1 4 7 13 -1 -1 -1 -1 -1 -1 -1
```

Sample Output 1:

```
1 14
```

Sample Input 2:

```
10 20 60 -1 -1 3 50 -1 -1 -1 -1
```

Sample Output 2:

```
3 60
```

```
public class Solution {

    private static Pair<Integer,Integer> maxMin=new 
Pair<Integer,Integer>(Integer.MAX_VALUE,Integer.MIN_VALUE);



	public static Pair<Integer, Integer> getMinAndMax(BinaryTreeNode<Integer> root) {

		//Your code goes here
```

```
         getMinAndMaxHelper(root);

        return maxMin;



             }



    private static void getMinAndMaxHelper(BinaryTreeNode<Integer> root)

    {

        if (root==null)

        {

            return;

        }



        int rootData=root.data;

        int maxVal=maxMin.maximum;

        if (rootData>maxVal)

        {

            maxMin.maximum=root.data;

        }



        int minVal=maxMin.minimum;

        if (rootData<minVal)

        {

            maxMin.minimum=root.data;

        }

        getMinAndMaxHelper(root.left);

        getMinAndMaxHelper(root.right);

             }



}
```