Mirror Binary Tree

Send Feedback

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

Example:

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);




            }


}
```

Diameter Of Binary Tree

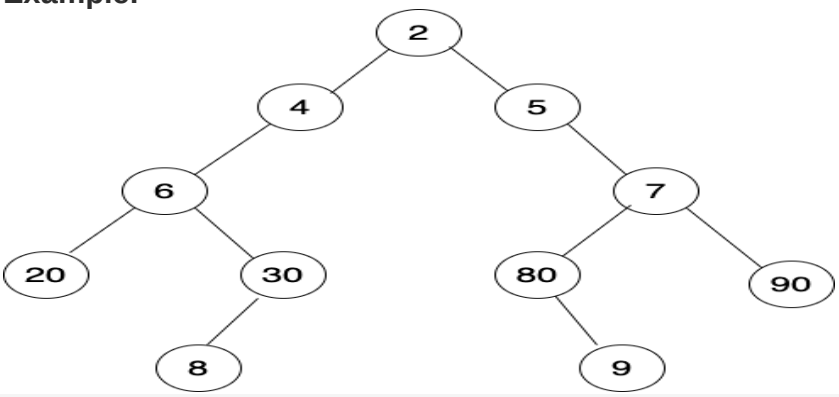
Send Feedback

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
```

```
/*


        Following is the structure used to represent the Binary Tree Node


        class BinaryTreeNode<T> {
                T data;
                BinaryTreeNode<T> left;
                BinaryTreeNode<T> right;


                public BinaryTreeNode(T data) {
                        this.data = data;
                        this.left = null;
                        this.right = null;
                }
        }


*/


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;
        }
    }

}
```

Print Levelwise

Send Feedback

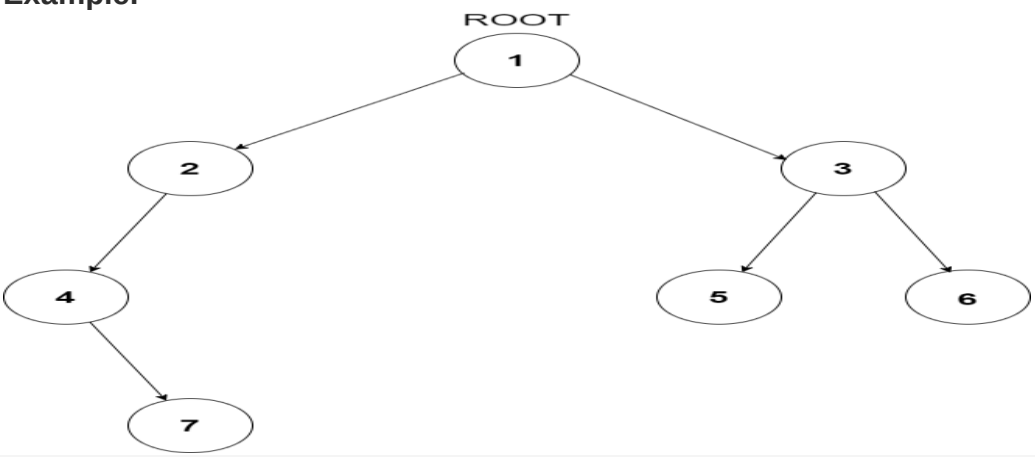
For a given a Binary Tree of type integer, print the complete information of every node, when traversed in a level-order fashion.

To print the information of a node with data D, you need to follow the exact format :

D:L:X,R:Y

Where D is the data of a node present in the binary tree.
X and Y are the values of the left(L) and right(R) child of the node.
Print -1 if the child doesn't exist.

Example:



For the above depicted Binary Tree, the level order travel will be printed as followed:

1:L:2,R:3

2:L:4:,R:-1
3:L:5:,R:6
4:L:-1:,R:7
5:L:-1:,R:-1
6:L:-1:,R:-1
7:L:-1:,R:-1

Note: There is no space in between while printing the information for each 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:

Information of all the nodes in the Binary Tree will be printed on a different line where each node will follow a format of D:L:X,R:Y, without any spaces in between.

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

Sample Output 1:

8:L:3,R:10
3:L:1,R:6
10:L:-1,R:14
1:L:-1,R:-1
6:L:4,R:7
14:L:13,R:-1
4:L:-1,R:-1
7:L:-1,R:-1
13:L:-1,R:-1

Sample Input 2:

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

Sample Output 2:

1:L:2,R:3
2:L:4,R:5
3:L:6,R:7
4:L:-1,R:-1
5:L:-1,R:-1
6:L:-1,R:-1
7:L:-1,R:-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;

        }


}
```

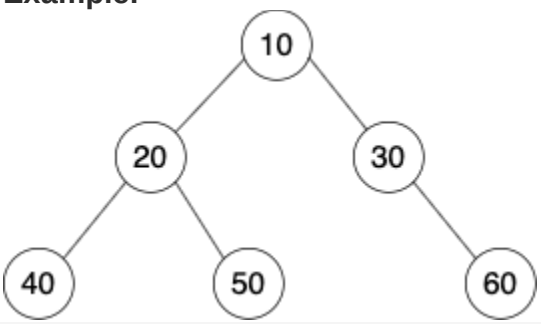
Create & Insert Duplicate Node

Send Feedback

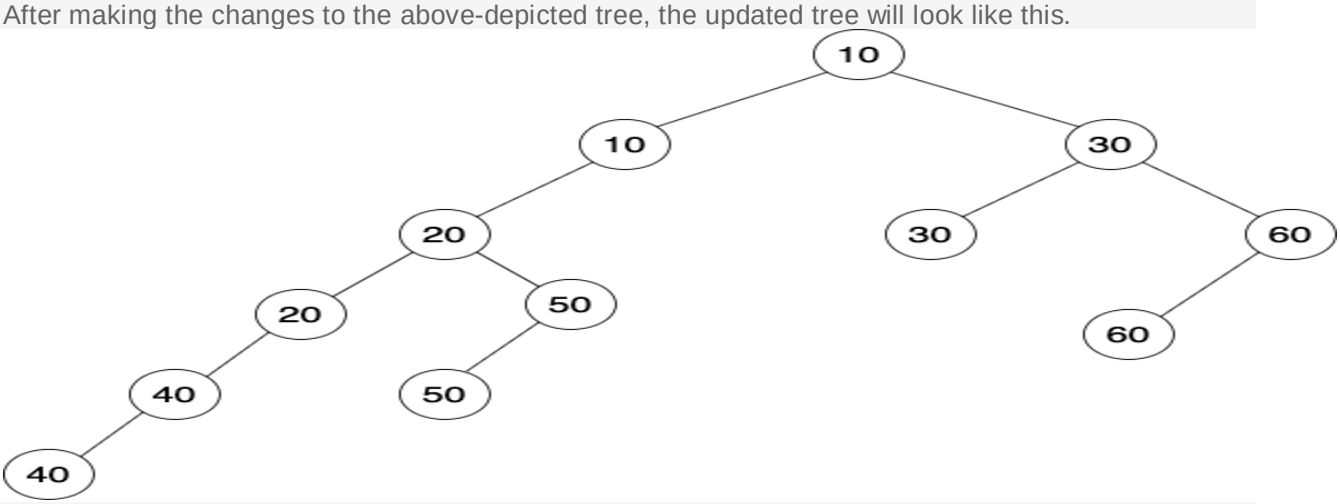
For a given a Binary Tree of type integer, duplicate every node of the tree and attach it to the left of itself.

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);



        }


}
```

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 -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)
```

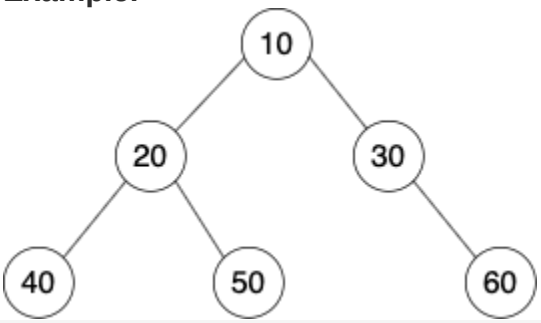
```
    {
        maxMin.minimum=root.data;
    }
    getMinAndMaxHelper(root.left);
    getMinAndMaxHelper(root.right);
        }

}
```

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)
```

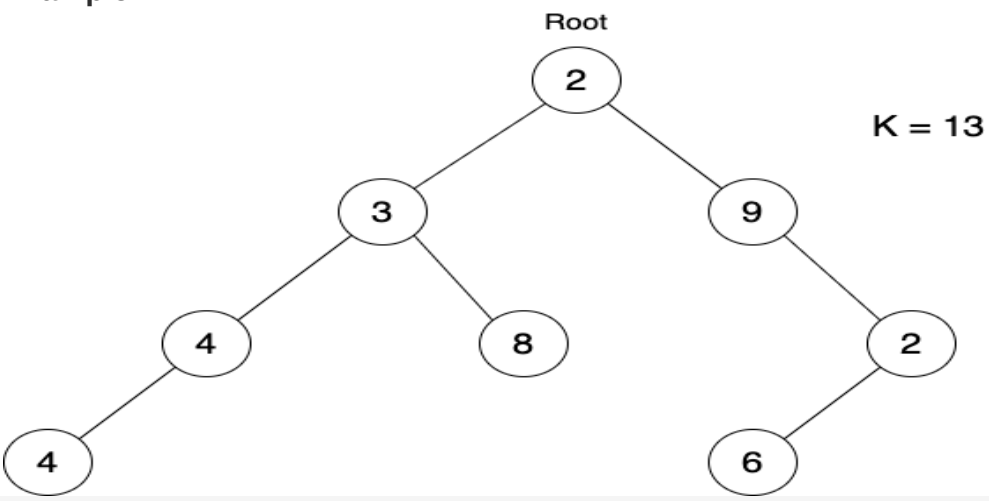
```
            nodesToPrint.add(front.left);
          if (front.right!=null)
            nodesToPrint.add(front.right);
        }
      }
        }

}
```

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);

    }

}
```

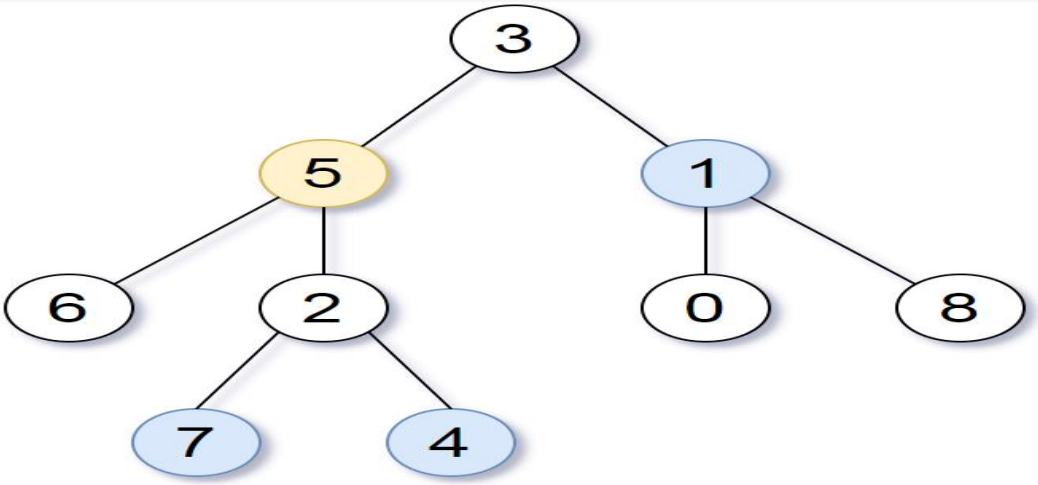
Print nodes at distance k from node

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);

    }

}
```