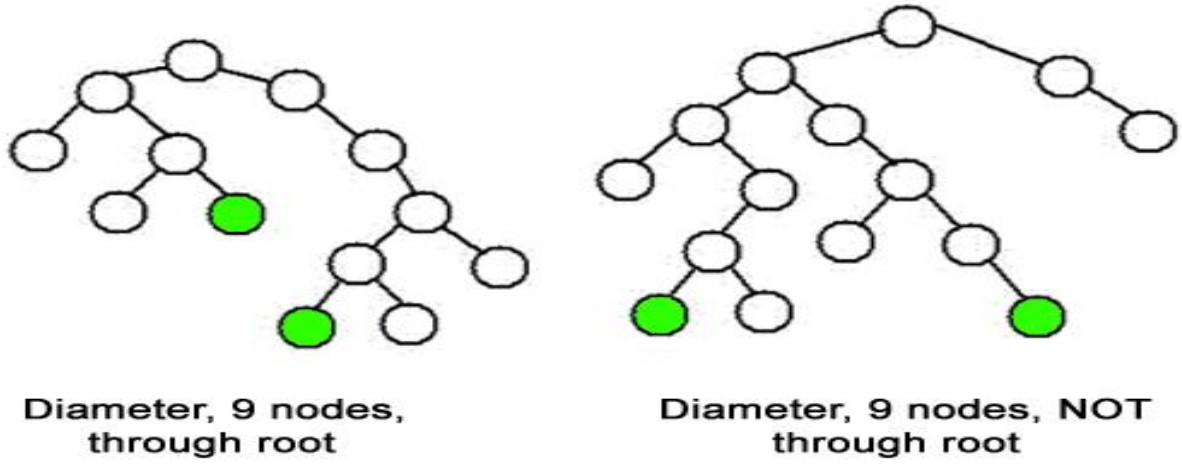
Question:

Given a Binary Tree, find the diameter of it.

The diameter of a tree is the number of nodes on the longest path between two end nodes in the tree. The diagram below shows two trees each with diameter nine, the leaves that form the ends of a 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).



Example 1:

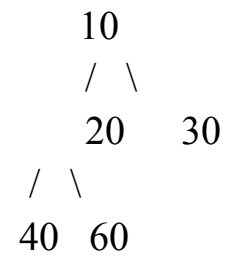
Input:

```
      1
    /  \
  2    3
```

Output: 3

Example 2:

Input:



Output: 4

Your Task:

You need to complete the function diameter() that takes root as parameter and returns the diameter.

Expected Time Complexity: O(N).

Expected Auxiliary Space: O(Height of the Tree).

Constraints:

1 <= Number of nodes <= 10000

1 <= Data of a node <= 1000

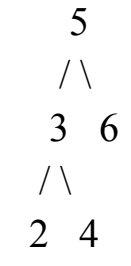
##########

Question :

Given a binary tree, the task is to create a new binary tree which is a mirror image of the given binary tree.

Examples:

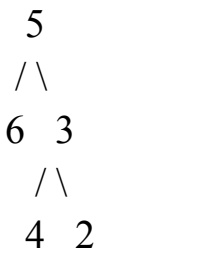
Input:



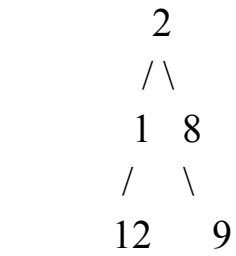
Output:

Inorder of original tree: 2 3 4 5 6
Inorder of mirror tree: 6 5 4 3 2

Mirror tree will be:



Input:

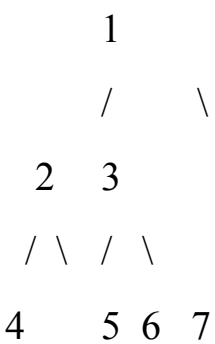


Output:
Inorder of original tree: 12 1 2 8 9
Inorder of mirror tree: 9 8 2 1 12

##########

Question :

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

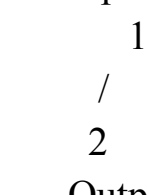


Top view will be: 4 2 1 3 7

Note: Print from leftmost node to rightmost node.

Example 1:

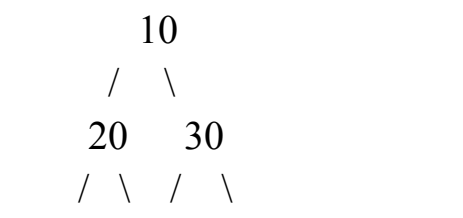
Input:



Output: 2 1 3

Example 2:

Input:



Output: 40 20 10 30 100

Your Task:

Since this is a function problem. You don't have to take input. Just complete the function printTopView() that takes the root node as parameter and prints the top view. Print endline after end of printing the top view.

Expected Time Complexity: O(N)

Expected Auxiliary Space: O(N).

Constraints:

1 <= N <= 105

1 <= Node Data <= 105

##

Question:

Given a binary tree, find its height.â€‹

Example 1:

Input:

```
    1
  /  \
 2    3
```

Output: 2

Example 2:

Input:

```
  2
   \
    1
   /
 3
```

Output: 3

Your Task:

You don't need to read input or print anything. Your task is to complete the function height() which takes the root node of the tree as input parameter and returns an integer denoting the height of the tree. If the tree is empty, return 0.

Expected Time Complexity: O(N)

Expected Auxiliary Space: O(N)

Constraints:

1 <= Number of nodes <= 10^5

1 <= Data of a node <= 10^5

##

Question:

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 :

Output: 3 2 1

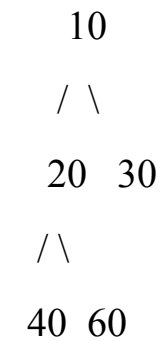
Explanation:

Traversing level 1 : 3 2

Traversing level 0 : 1

Example 2:

Input :



Output: 40 60 20 30 10

Explanation:

Traversing level 2 : 40 60

Traversing level 1 : 20 30

Traversing level 0 : 10

Your Task:

You don't need to read input or print anything. Complete the function reverseLevelOrder() which takes the root of the tree as input parameter and returns a list containing the reverse level order traversal of the given tree.

Expected Time Complexity: O(N)

Expected Auxiliary Space: O(N)

Constraints:

$1 \leq N \leq 10^4$

##

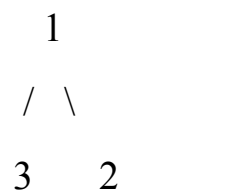
Question:

Given a binary tree, find its level order traversal.

Level order traversal of a tree is breadth-first traversal for the tree.

Example 1:

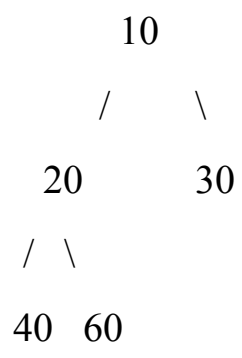
Input:



Output:1 3 2

Example 2:

Input:



Output:10 20 30 40 60 N N

Your Task:

You don't have to take any input. Complete the function levelOrder() that takes the root node as input parameter and returns a list of integers containing the level order traversal of the given Binary Tree.

Expected Time Complexity: O(N)

Expected Auxiliary Space: O(N)

Constraints:

1 <= Number of nodes<= 104

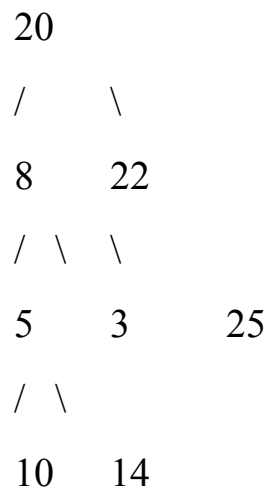
1 <= Data of a node <= 104

##########

Question:

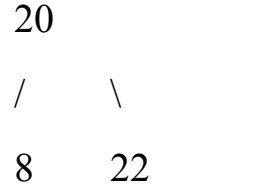
Given a binary tree, print the bottom view from left to right.

A node is included in the bottom view if it can be seen when we look at the tree from bottom.



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.



/ \ / \

5 3 4 25

/ \

10 14

For the above tree the output should be 5 10 4 14 25.

Example 1:

Input:

1

/ \

3 2

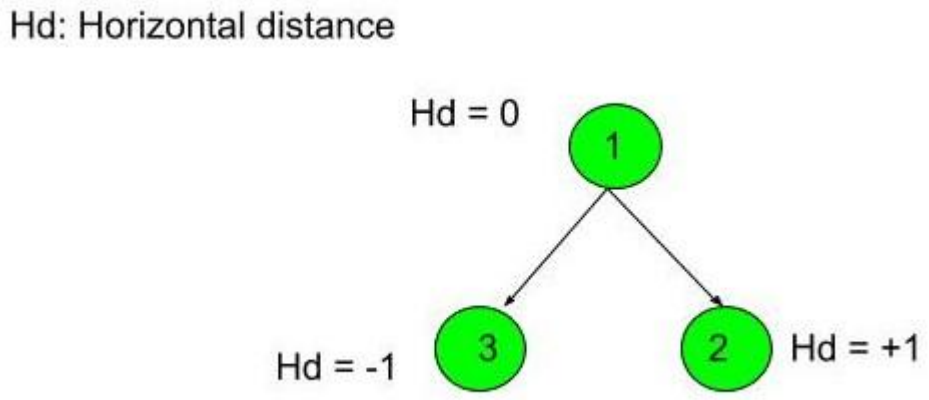
Output: 3 1 2

Explanation:

First case represents a tree with 3 nodes

and 2 edges where root is 1, left child of

1 is 3 and the right child of 1 is 2.

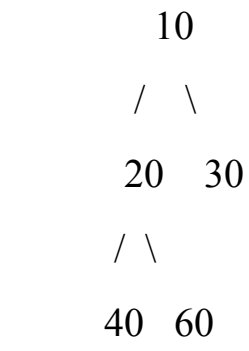


Thus nodes of the binary tree will be

printed as such 3 1 2.

Example 2:

Input:



Output: 40 20 60 30

Your Task:

This is a functional problem, you don't need to care about input, just complete the function bottomView() which takes the root node of the tree as input and returns an array containing the bottom view of the given tree.

Expected Time Complexity: O(N).

Expected Auxiliary Space: O(N).

Constraints:

1 <= Number of nodes <= 105

1 <= Data of a node <= 105

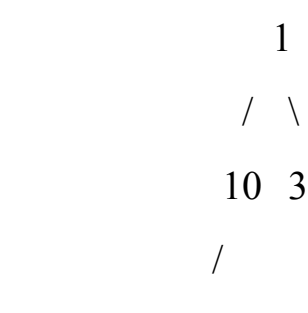
##########

Question:

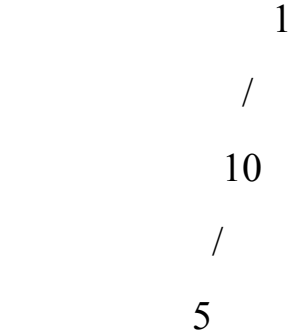
Given a binary tree, find if it is height balanced or not.

A tree is height balanced if the difference between heights of left and right subtrees is not more than one for all nodes of the tree.

A height balanced tree

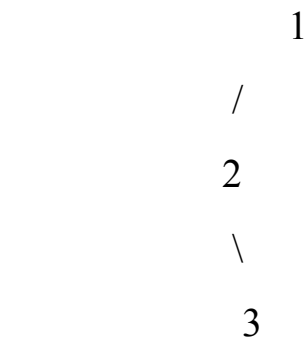


An unbalanced tree



Example 1:

Input:



Output: 0

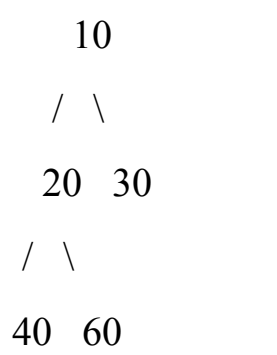
Explanation: The max difference in height

of left subtree and right subtree is 2,

which is greater than 1. Hence unbalanced

Example 2:

Input:



Output: 1

Explanation: The max difference in height

The left subtree and right subtree is 1.

Hence balanced.

Your Task:

You don't need to take input. Just complete the function isBalanced() that takes the root node as parameter and returns true, if the tree is balanced else returns false.

Constraints:

1 <= Number of nodes <= 105

0 <= Data of a node <= 106

Expected time complexity: O(N)

Expected auxiliary space: O(h) , where h = height of tree

##

Question:

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

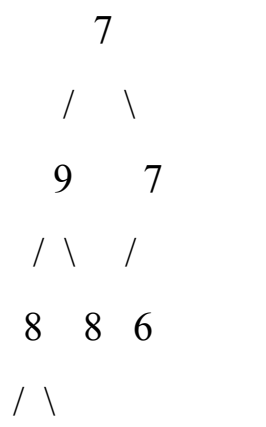
Example 1:

Input:

Output: 3 1 2

Example 2:

Input:



Output: 7 7 9 8 8 6 9 10

Your Task:

You don't need to read input or print anything. Your task is to complete the function zigZagTraversal() which takes the root node of the Binary Tree as its input and returns a list containing the node values as they appear in the Zig-Zag Level-Order Traversal of the Tree.

Expected Time Complexity: O(N).

Expected Auxiliary Space: O(N).

Constraints:

1 <= N <= 104

##

Question:

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

Example 1:

Input:

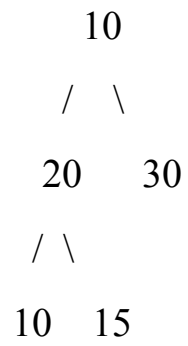
Output: 1

Explanation:

Leaves 2 and 3 are at the same level.

Example 2:

Input:



Output: 0

Explanation:

Leaves 10, 15 and 30 are not at the same level.

Your Task:

You don't need to read input or print anything. Complete the function check() which takes the root node as input parameter and returns true/false depending on whether all the leaf nodes are at the same level or not.

Expected Time Complexity: O(N)

Expected Auxiliary Space: O(height of tree)

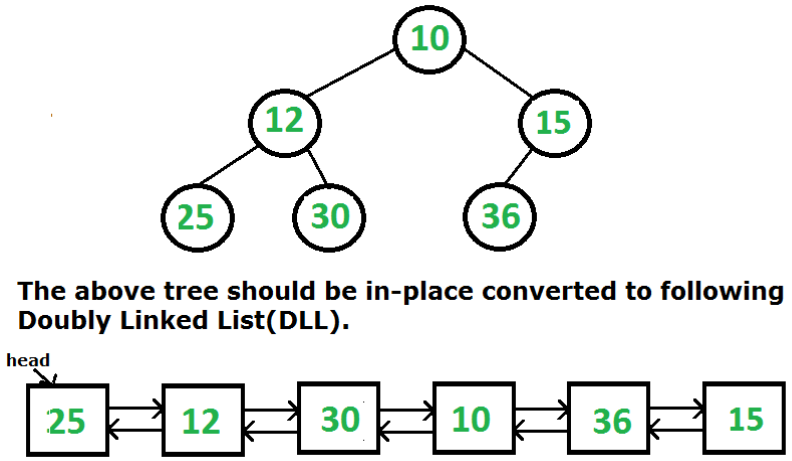
Constraints:

$1 \leq N \leq 10^3$

##########

Question:

Given a Binary Tree (BT), convert it to a Doubly Linked List(DLL) In-Place. The left and right pointers in nodes are to be used as previous and next pointers respectively in converted DLL. The order of nodes in DLL must be the same as Inorder of the given Binary Tree. The first node of Inorder traversal (leftmost node in BT) must be the head node of the DLL.



Example 1:

Input:

```
    1
  /  \
 3    2
```

Output:

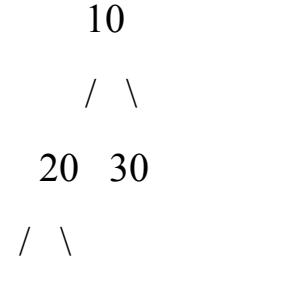
3 1 2

2 1 3

Explanation: DLL would be 3<=>1<=>2

Example 2:

Input:



Output:

40 20 60 10 30

30 10 60 20 40

Explanation: DLL would be

40<=>20<=>60<=>10<=>30.

Your Task:

You don't have to take input. Complete the function bToDLL() that takes root node of the tree as a parameter and returns the head of DLL . The driver code prints the DLL both ways.

Expected Time Complexity: O(N).

Expected Auxiliary Space: O(H).

Note: H is the height of the tree and this space is used implicitly for recursion stack.

Constraints:

1 <= Number of nodes <= 1000

1 <= Data of a node <= 1000

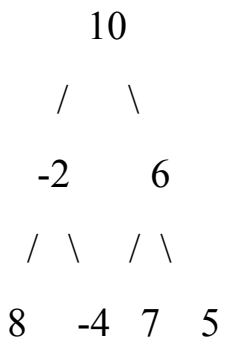
##

Question:

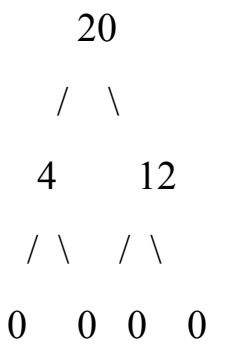
Given a Binary Tree of size N , where each node can have positive or negative values. Convert this to a tree where each node contains the sum of the left and right subtrees of the original tree. The values of leaf nodes are changed to 0.

Example 1:

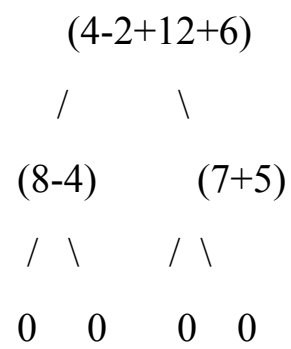
Input:



Output:



Explanation:



Your Task:

You don't need to read input or print anything. Complete the function toSumTree() which takes the root node as input parameter and modifies the given tree in-place.

Note: If you click on Compile and Test the output will be the in-order traversal of the modified tree.

Expected Time Complexity: O(N)

Expected Auxiliary Space: O(height of tree)

Constraints:

$1 \leq N \leq 104$

##########

Question:

Construct a binary tree from a string consisting of parentheses and integers. The whole input represents a binary tree. It contains an integer followed by zero, one or two pairs of parenthesis. The integer represents the root’s value and a pair of parentheses contains a child binary tree with the same structure. Always start to construct the left child node of the parent first if it exists.

Examples:

Input : "1(2)(3)"

Output : 1 2 3

Explanation :

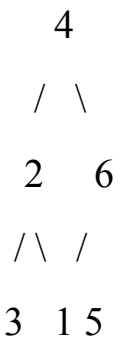
```
    1
   / \
  2   3
```

Explanation: first pair of parentheses contains
left subtree and second one contains the right
subtree. Preorder of above tree is "1 2 3".

Input : "4(2(3)(1))(6(5))"

Output : 4 2 3 1 6 5

Explanation :



##

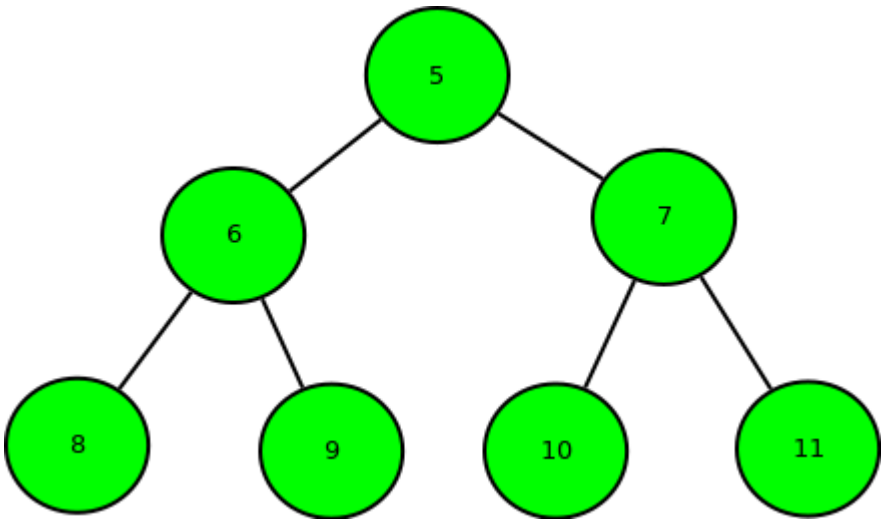
Given the array representation of Complete Binary Tree i.e, if index i is the parent, index 2*i + 1 is the left child and index 2*i + 2 is the right child. The task is to find the minimum number of swap required to convert it into Binary Search Tree.

Examples:

Input : arr[] = { 5, 6, 7, 8, 9, 10, 11 }

Output : 3

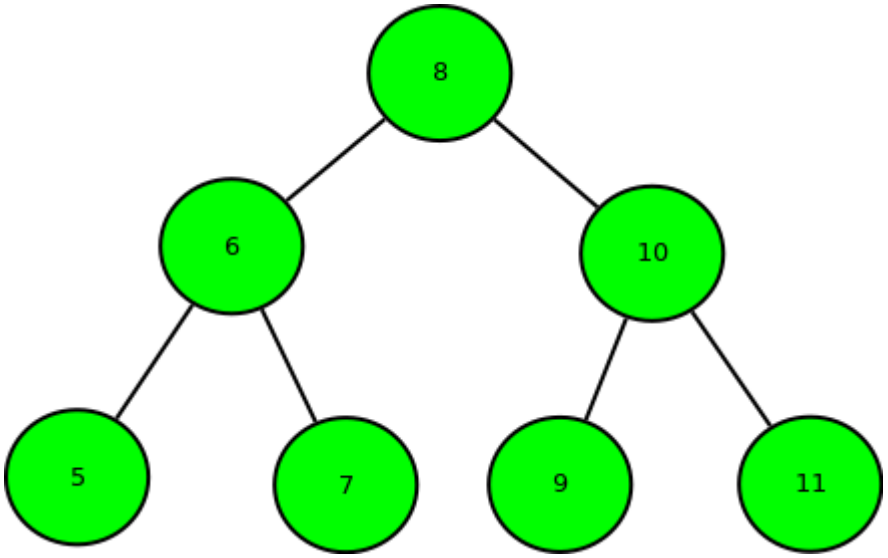
Binary tree of the given array:



Swap 1: Swap node 8 with node 5.

Swap 2: Swap node 9 with node 10.

Swap 3: Swap node 10 with node 7.

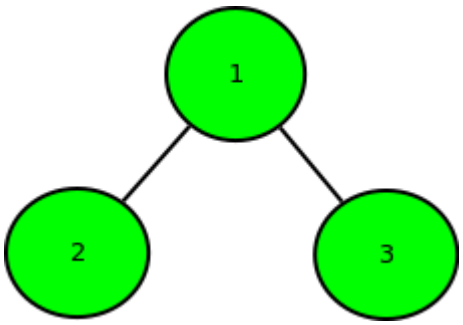


So, minimum 3 swaps are required.

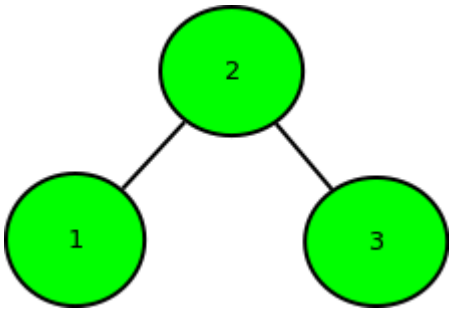
Input : arr[] = { 1, 2, 3 }

Output : 1

Binary tree of the given array:



After swapping node 1 with node 2.

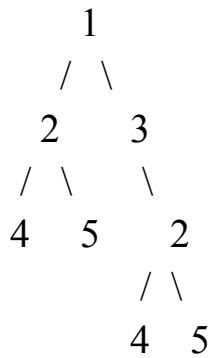


So, only 1 swap required.

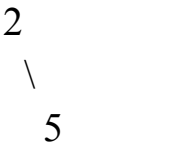
##
Given a binary tree, find out whether it contains a duplicate sub-tree of size two or more, or not.

Example 1 :

Input :



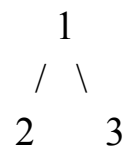
Output : 1
Explanation :



is the duplicate sub-tree.

Example 2 :

Input :



Output: 0
Explanation: There is no duplicate sub-tree
in the given binary tree.

Your Task:

You don't need to read input or print anything. Your task is to complete the function dupSub() which takes root of the tree as the only arguement and returns 1 if the binary tree contains a duplicate sub-tree of size two or more, else 0.

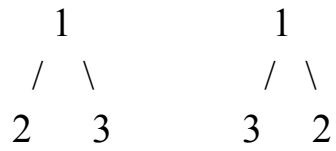
Note: Two same leaf nodes are not considered as subtree as size of a leaf node is one.

Constraints:

1<=length of string<=100

##
Given two n-ary tree's the task is to check if they are mirror of each other or not.

Example



Output: 1

Output: 0

Note: you may assume that root of both the given tree as 1.

Input:
The first line of input contains an integer T denoting the no of test cases. Then T test cases follow. The first line of each test case contains two space separated values n and e denoting the no of nodes and edges respectively. Then in the next line two lines are 2*e space separated values u,v denoting an edge from u to v for the both trees .

Output:
For each test case in a new line print 1 if both the trees are mirrors of each other else print 0.

Constraints:
1<=T<=20
1<=n<=15
1<=e<=20

Example:
Input:

2
3 2
1 2 1 3
1 3 1 2
3 2
1 2 1 3
1 2 1 3
Output:
1
0