**Delete a node from BST**

**Medium**

Given a Binary Search Tree and a node value X. Delete the node with the given value X from the BST. If no node with value x exists, then do not make any change.

**Example 1:**

**Input:**

2

  / \

  1 3

X = 12

**Output:** 1 2 3

**Explanation:** In the given input there

is no node with value 12 , so the tree

will remain same.

**Example 2:**

**Input:**

            1

            \

          2

         \

        8

               /    \

         5    11

           /  \    / \

      4    7  9   12

X = 9

**Output:** 1 2 4 5 7 8 11 12

**Explanation:** In the given input tree after

deleting 9 will be

            1

            \

          2

          \

          8

               /  \

         5    11

               /  \     \

      4    7     12

**Expected Time Complexity:**O(Height of the BST).  
**Expected Auxiliary Space:**O(Height of the BST).

**Constraints:**  
1 <= N <= 105

**Company Tags**

[**Accolite**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Accolite) [**Amazon**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Amazon) [**Samsung**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Samsung) [**Adobe**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Adobe) [**Qualcomm**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Qualcomm)

//{ Driver Code Starts

// Initial Template for Java

// Initial Template for Java

import java.util.LinkedList;

import java.util.Queue;

import java.io.\*;

import java.util.\*;

class Node {

int data;

Node left;

Node right;

Node(int data) {

this.data = data;

left = null;

right = null;

}

}

class CodingMaaxima {

static Node buildTree(String str) {

if (str.length() == 0 || str.charAt(0) == 'N') {

return null;

}

String ip[] = str.split(" ");

// Create the root of the tree

Node root = new Node(Integer.parseInt(ip[0]));

// Push the root to the queue

Queue<Node> queue = new LinkedList<>();

queue.add(root);

// Starting from the second element

int i = 1;

while (queue.size() > 0 && i < ip.length) {

// Get and remove the front of the queue

Node currNode = queue.peek();

queue.remove();

// Get the current node's value from the string

String currVal = ip[i];

// If the left child is not null

if (!currVal.equals("N")) {

// Create the left child for the current node

currNode.left = new Node(Integer.parseInt(currVal));

// Push it to the queue

queue.add(currNode.left);

}

// For the right child

i++;

if (i >= ip.length) break;

currVal = ip[i];

// If the right child is not null

if (!currVal.equals("N")) {

// Create the right child for the current node

currNode.right = new Node(Integer.parseInt(currVal));

// Push it to the queue

queue.add(currNode.right);

}

i++;

}

return root;

}

static void printInorder(Node root) {

if (root == null) return;

printInorder(root.left);

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

printInorder(root.right);

}

public static void main(String[] args) throws Exception {

BufferedReader br =

new BufferedReader(new InputStreamReader(System.in));

int t = Integer.parseInt(br.readLine());

while (t > 0) {

String s = br.readLine();

int x = Integer.parseInt(br.readLine());

Node root = buildTree(s);

Tree g = new Tree();

root = g.deleteNode(root, x);

printInorder(root);

System.out.println();

t--;

}

}

}

// } Driver Code Ends

// User function Template for Java

class Tree {

private static Node minNode(Node head){

while(head.left!=null){

head=head.left;

}

return head;

}

public static Node deleteNode(Node root, int x) {

if(root==null)

return null;

if(root.data>x){

root.left=deleteNode(root.left,x);

}

else if(root.data<x){

root.right=deleteNode(root.right, x);

}

else{

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

return null;

else

if(root.left==null)

return root.right;

else if(root.right==null)

return root.left;

Node temp = minNode(root.right);

root.data = temp.data;

root.right = deleteNode(root.right, temp.data);

}

return root;

}

}