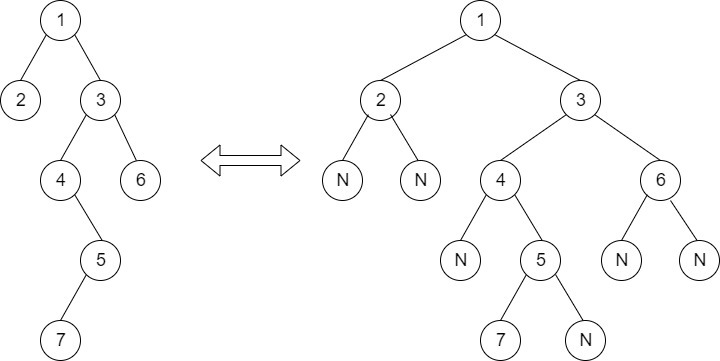
Given a Binary Search Tree (BST) and a node value. Delete the node with the given value from the BST. If no node with value x exists, then do not make any change.

**Input:**  
The first line of input contains the number of test cases T. For each test case, there will two lines. First line of input is a string representing the tree as described below:

1. The values in the string are in the order of level order traversal of the tree where, numbers denote node values, and a character “N” denotes NULL child.
2. For example:  
      
   For the above tree, the string will be: 1 2 3 N N 4 6 N 5 N N 7 N
3. Second-line is an integer represents X

**Output:**  
For each test case in a new line, print the inorder traversal of the modified BST.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **deleteNode()** which takes two arguments. The first being the root of the tree, and an integer **'X'** denoting the node value to be deleted from the BST. Return the root of the BST after deleting the node with value X. Do not make any update if there's no node with value X present in the BST.

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

**Constraints:**  
1 <= T <= 10  
1 <= N <= 100000

**Example:  
Input:**  
2  
2 1 3 N N N N  
12  
1 N 2 N 8 5 11 4 7 9 12  
9  
**Output:**  
1 2 3  
1 2 4 5 7 8 11 12

**Explanation:**  
**Testcase 1:**In the given input there is no node with value 12 , so the tree will remain same.  
**Testcase 2:** In the given input tree after deleting 9 will be  
                        1  
                  /               \  
                N                2  
                              /           \  
                            N            8  
                                     /            \  
                                   5              11  
                              /          \              \  
                            4           7             12

**Note:** The **Input/Output** format and **Example**are given are used for the system's internal purpose, and should be used by a user for **Expected Output** only. As it is a function problem, hence a user should not read any input from the stdin/console. The task is to complete the function specified, and not to write the full code.

\*\* For More Input/Output Examples Use ['Expected Output'](https://practice.geeksforgeeks.org/problems/delete-a-node-from-bst/1/?track=amazon-bst&batchId=192#ExpectOP) option \*\*

// { Driver Code Starts

#include <bits/stdc++.h>

using namespace std;

#define MAX\_HEIGHT 100000

// Tree Node

struct Node {

int data;

Node \*left;

Node \*right;

Node(int val) {

data = val;

left = right = NULL;

}

};

struct Node\* deleteNode(struct Node\* root, int key);

// Function to Build Tree

Node\* buildTree(string str)

{

// Corner Case

if(str.length() == 0 || str[0] == 'N')

return NULL;

// Creating vector of strings from input

// string after spliting by space

vector<string> ip;

istringstream iss(str);

for(string str; iss >> str; )

ip.push\_back(str);

// Create the root of the tree

Node\* root = new Node(stoi(ip[0]));

// Push the root to the queue

queue<Node\*> queue;

queue.push(root);

// Starting from the second element

int i = 1;

while(!queue.empty() && i < ip.size()) {

// Get and remove the front of the queue

Node\* currNode = queue.front();

queue.pop();

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

string currVal = ip[i];

// If the left child is not null

if(currVal != "N") {

// Create the left child for the current node

currNode->left = new Node(stoi(currVal));

// Push it to the queue

queue.push(currNode->left);

}

// For the right child

i++;

if(i >= ip.size())

break;

currVal = ip[i];

// If the right child is not null

if(currVal != "N") {

// Create the right child for the current node

currNode->right = new Node(stoi(currVal));

// Push it to the queue

queue.push(currNode->right);

}

i++;

}

return root;

}

void inorder(Node \*root, vector<int> &v)

{

if(root==NULL)

return;

inorder(root->left, v);

v.push\_back(root->data);

inorder(root->right, v);

}

int main() {

int t;

string tc;

getline(cin, tc);

t=stoi(tc);

while(t--)

{

string s;

getline(cin, s);

Node\* root1 = buildTree(s);

getline(cin, s);

int k = stoi(s);

root1 = deleteNode(root1, k);

vector<int> v;

inorder(root1, v);

for(auto i:v)

cout << i << " ";

cout << endl;

}

return 0;

}// } Driver Code Ends

/\* The structure of a BST Node is as follows:

struct Node {

int data;

Node \*left;

Node \*right;

Node(int val) {

data = val;

left = right = NULL;

}

};

\*/

// Return the root of the modified BST after deleting the node with value X

Node \*deleteNode(Node \*root, int x)

{

if(root==NULL)

return root;

else 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)

{

delete root;

root = NULL;

}

else if(root->left==NULL)

{

Node\* temp=root;

root=root->right;

temp->right=NULL;

//delete temp;

}

else if(root->right==NULL)

{

Node\* temp=root;

root=root->left;

temp->left=NULL;

//delete temp;

}

else

{

//when we have elements on both sides

Node\* temp=root->left;

while(temp->right!=NULL)

{

temp=temp->right;

}

int leftmax=temp->data;

root->data=leftmax;

root->left=deleteNode(root->left,leftmax);

}

}

return root;

}