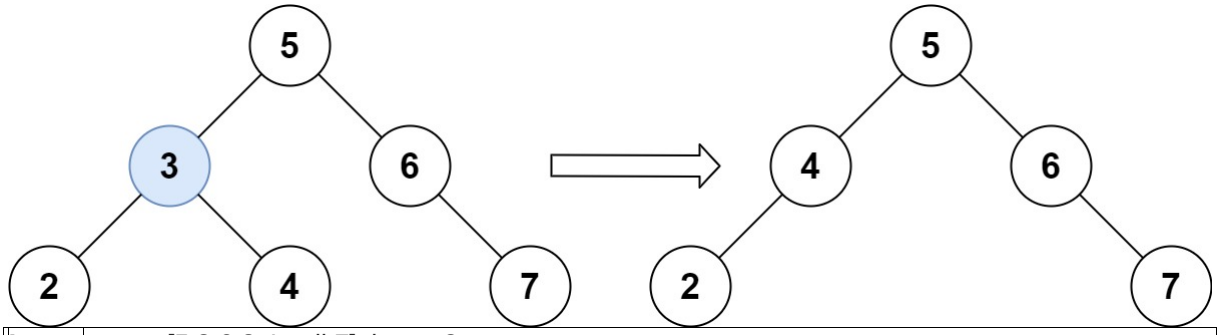
Delete Node in a BST
Medium
7.3K
185
Companies

Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return *the **root node reference** (possibly updated) of the BST*.

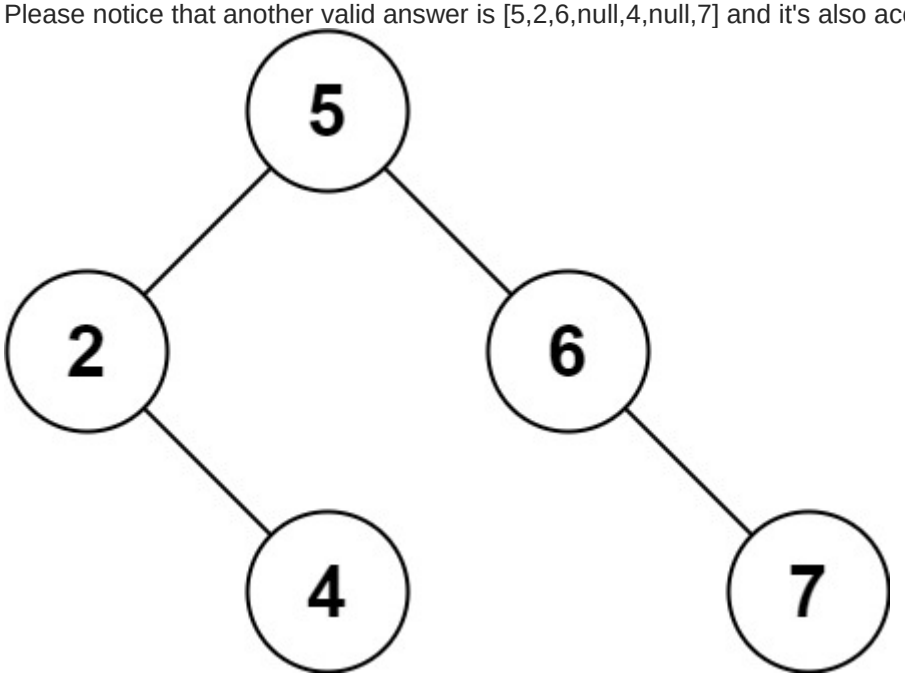
Basically, the deletion can be divided into two stages:

1. Search for a node to remove.
2. If the node is found, delete the node.

Example 1:



Input: root = [5,3,6,2,4,null,7], key = 3
Output: [5,4,6,2,null,null,7]
Explanation: Given key to delete is 3. So we find the node with value 3 and delete it.
One valid answer is [5,4,6,2,null,null,7], shown in the above BST.
Please notice that another valid answer is [5,2,6,null,4,null,7] and it's also accepted.



Example 2:

Input: root = [5,3,6,2,4,null,7], key = 0
Output: [5,3,6,2,4,null,7]
Explanation: The tree does not contain a node with value = 0.

Example 3:

Input: root = [], key = 0
Output: []

Constraints:

- The number of nodes in the tree is in the range [0, 10^4].
- -10^5 <= Node.val <= 10^5
- Each node has a **unique** value.
- root is a valid binary search tree.
- -10^5 <= key <= 10^5

Question Link- https://leetcode.com/problems/delete-node-in-a-bst/description/

Code-

```
class Solution {
public:
    TreeNode* inordersuccessor(TreeNode *root)
    {
        if(root->right!=NULL){
            root=root->right;
            while(root->left!=NULL){
                root=root->left;
            }
        }
        return root;
    }

    TreeNode* deleteNode(TreeNode* root, int key) {
        if(root==NULL){
            return root;
        }
        if(root->val<key){
            root->right=deleteNode(root->right,key);
        }
        else if(root->val>key){
            root->left=deleteNode(root->left,key);
        }
        else
        {
```

```
            if(root->left==NULL){
                TreeNode* temp=root->right;
                delete root;
                return temp;
            }
            else if(root->right==NULL){
                TreeNode* temp=root->left;
                delete root;
                return temp;
            }
            else{
                TreeNode* suc=inordersuccessor(root);
                root->val=suc->val;
                root->right=deleteNode(root->right,suc->val);
            }
        }
        return root;
    }
};
```