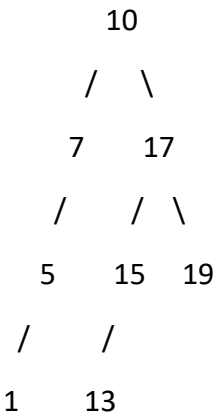
Delete Node from BST LeetCode

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:

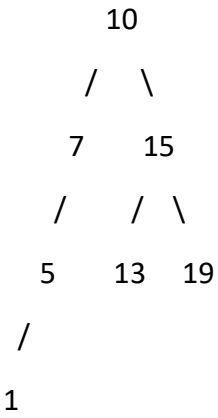
1. Search for a node to remove.
2. If the node is found, delete the node.

Example:



DeleteNode: 17

Output:



Approach 1: Function to delete a node with the minimum value in the BST (Inorder Predecessor)

1. **Function Purpose:** This approach is used to delete a node with a specific key from the BST.
2. **Base Case:** The function checks if the root of the BST is null. If the tree is empty, it returns null, indicating that there's nothing to delete.

3. **Node Deletion:** When the root node's value matches the key, there are several cases to consider:
    - If the node to be deleted is a leaf node (has no children), it is simply deleted, and the function returns null.
    - If the node has a left child but no right child, it is replaced with its left child.
    - If the node has a right child but no left child, it is replaced with its right child.
    - If the node has both left and right children, it is replaced with the largest value from its left subtree (inorder predecessor).
4. **Recursion:** The function continues to recursively delete the node in the left or right subtree, depending on whether the key is smaller or larger than the current node's value.
5. **Time Complexity: The time complexity of this approach is O(H), where H is the height of the BST. In the worst case, when the tree is skewed, H can be equal to N, the number of nodes in the tree.**
6. **Space Complexity: The space complexity is also O(H) due to the recursion depth.**

Approach 2: Function to delete a node with the maximum value in the BST (Inorder Successor)

1. **Function Purpose:** This approach is used to delete a node with a specific key from the BST.
2. **Base Case:** The function checks if the root of the BST is null. If the tree is empty, it returns null, indicating that there's nothing to delete.
3. **Node Deletion:** When the root node's value matches the key, there are several cases to consider:
    - If the node to be deleted is a leaf node (has no children), it is simply deleted, and the function returns null.
    - If the node has a left child but no right child, it is replaced with its left child.
    - If the node has a right child but no left child, it is replaced with its right child.
    - If the node has both left and right children, it is replaced with the smallest value from its right subtree (inorder successor).
4. **Recursion:** The function continues to recursively delete the node in the left or right subtree, depending on whether the key is smaller or larger than the current node's value.

5. **Time Complexity: The time complexity of this approach is O(H), where H is the height of the BST. In the worst case, when the tree is skewed, H can be equal to N, the number of nodes in the tree.**
6. **Space Complexity: The space complexity is also O(H) due to the recursion depth.**

Conclusion:

- Both approaches for deleting nodes in a BST are effective and have the same time and space complexity.
- The choice between the two approaches depends on the specific requirements of your application. The Inorder Predecessor Approach is used when deleting a node with a value of 15, while the Inorder Successor Approach is used when deleting a node with a value of 17 in the provided code.