Deletion operation is a little more complicated. we also provide c++ and java codes for your reference.

|  |
| --- |
| /\*\*  \* Definition for a binary tree node.  \* public class TreeNode {  \* int val;  \* TreeNode left;  \* TreeNode right;  \* TreeNode(int x) { val = x; }  \* }  \*/  class Solution {  /\*\*  \* findSuccesor - Helper function for finding successor  \* In BST, succesor of root is the leftmost child in root's right subtree  \*/  private TreeNode findSuccessor(TreeNode root) {  TreeNode cur = root.right;  while (cur != null && cur.left != null) {  cur = cur.left;  }  return cur;  }  public TreeNode deleteNode(TreeNode root, int key) {  // return null if root is null  if (root == null) {  return root;  }  // delete current node if root is the target node  if (root.val == key) {  // replace root with root->right if root->left is null  if (root.left == null) {  return root.right;  }  // replace root with root->left if root->right is null  if (root.right == null) {  return root.left;  }  // replace root with its successor if root has two children  TreeNode p = findSuccessor(root);  root.val = p.val;  root.right = deleteNode(root.right, p.val);  return root;  }  if (root.val < key) {  // find target in right subtree if root->val < key  root.right = deleteNode(root.right, key);  } else {  // find target in left subtree if root->val > key  root.left = deleteNode(root.left, key);  }  return root;  }  } |

We are going to discuss the time complexity and the space complexity as usual.

For the implementation above, similar to the recursion solution of the search operation, the time complexity is O(h) in the worst case. And according to the depth of recursion, the space complexity is also O(h) in the worst case.

We can also represent the complexity using the total number of nodes N. The time complexity and space complexity will be O(logN) in the best case but O(N) in the worse case.

If you can try to implement the strategy iteratively, you will reduce the space complexity to O(1).