```
class Solution {
  /*
  One step right and then always left
  */
  public int successor(TreeNode root) {
    root = root.right;
    while (root.left != null) root = root.left;
    return root.val;
  }

  /*
  One step left and then always right
  */
  public int predecessor(TreeNode root) {
    root = root.left;
    while (root.right != null) root = root.right;
    return root.val;
  }

  public TreeNode deleteNode(TreeNode root, int key) {
    if (root == null) return null;

    // delete from the right subtree
    if (key > root.val) root.right = deleteNode(root.right, key);
    // delete from the left subtree
    else if (key < root.val) root.left = deleteNode(root.left, key);
    // delete the current node
    else {
      // the node is a leaf
      if (root.left == null && root.right == null) root = null;
      // the node is not a leaf and has a right child
      else if (root.right != null) {
        root.val = successor(root);
        root.right = deleteNode(root.right, root.val);
      }
      // the node is not a leaf, has no right child, and has a left child
      else {
        root.val = predecessor(root);
        root.left = deleteNode(root.left, root.val);
      }
    }
    return root;
  }
}
```

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.

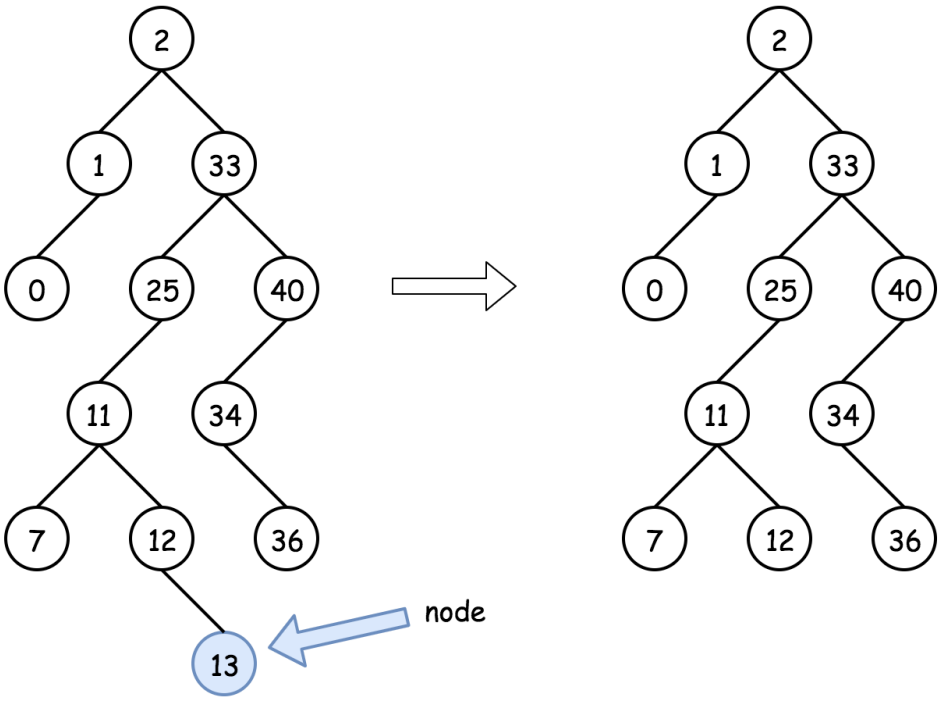
Note: Time complexity should be O(height of tree).

Approach 1: Recursion

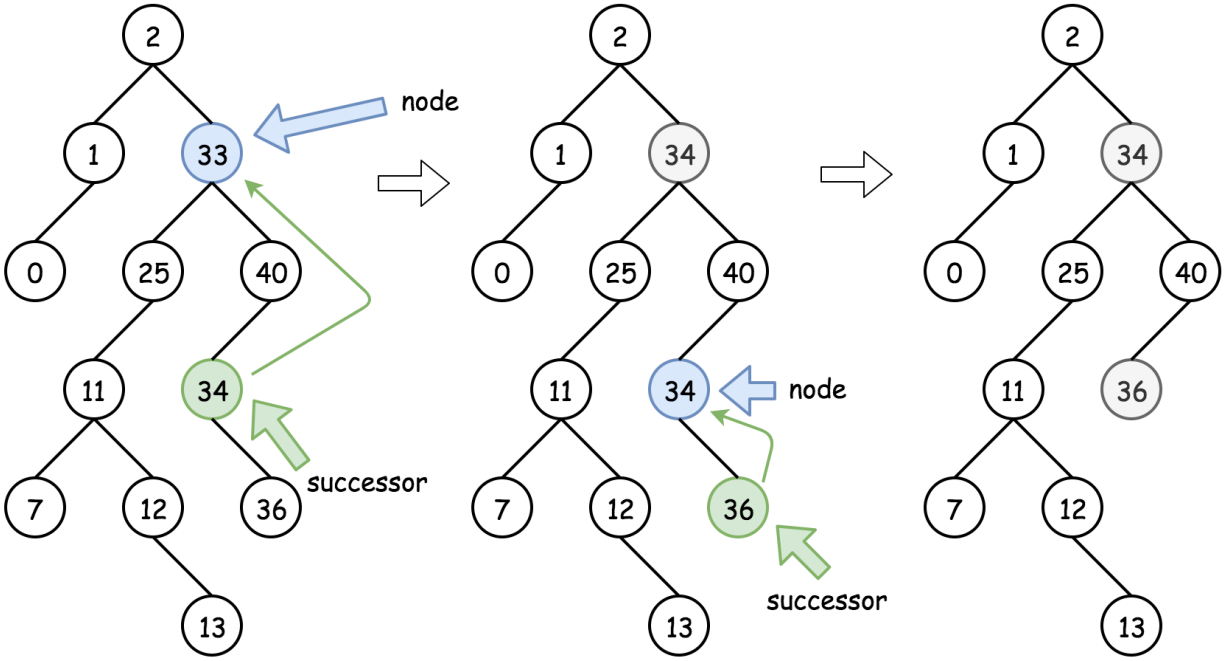
Intuition

There are three possible situations here :

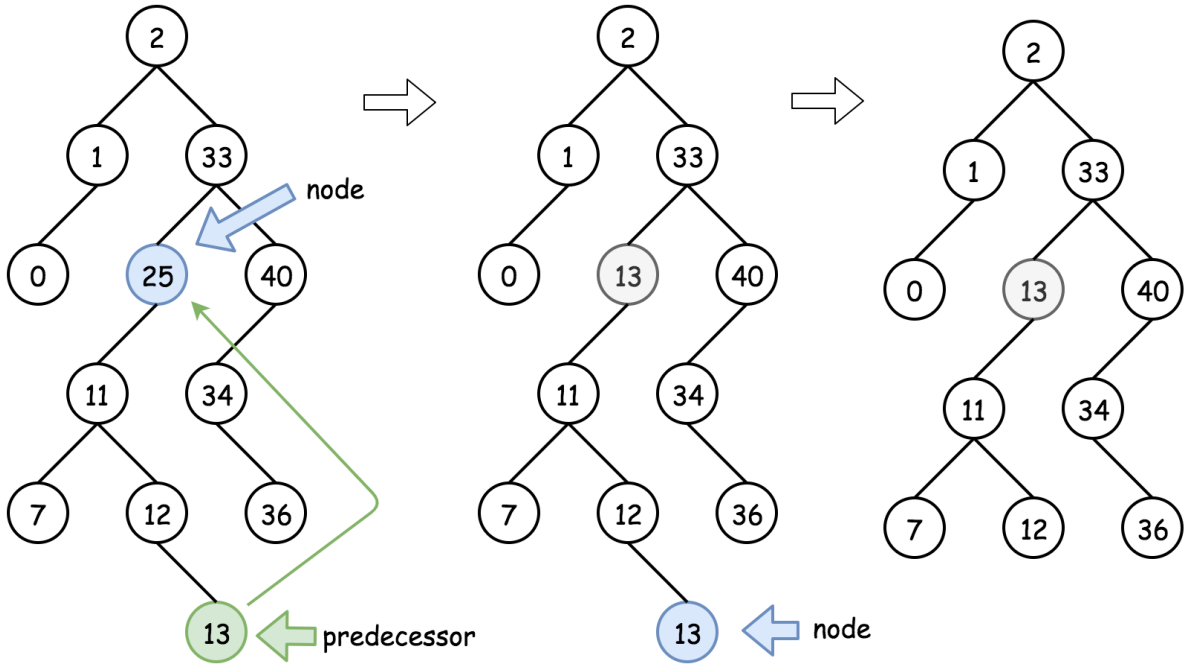
- Node is a leaf, and one could delete it straightforward : `node = null`.



- Node is not a leaf and has a right child. Then the node could be replaced by its *successor* which is somewhere lower in the right subtree. Then one could proceed down recursively to delete the successor.



- Node is not a leaf, has no right child and has a left child. That means that it's *successor* is somewhere upper in the tree but we don't want to go back. Let's use the *predecessor* here which is somewhere lower in the left subtree. The node could be replaced by its *predecessor* and then one could proceed down recursively to delete the predecessor.



Algorithm

- If `key > root.val` then delete the node to delete is in the right subtree `root.right = deleteNode(root.right, key)`.

- If `key < root.val` then delete the node to delete is in the left subtree `root.left = deleteNode(root.left, key)`.

- If `key == root.val` then the node to delete is right here. Let's do it :

    - If the node is a leaf, the delete process is straightforward : `root = null`.

    - If the node is not a leaf and has the right child, then replace the node value by a successor value `root.val = successor.val`, and then recursively delete the successor in the right subtree `root.right =`

`deleteNode(root.right, root.val)`.

    - If the node is not a leaf and has only the left child, then replace the node value by a predecessor value `root.val = predecessor.val`, and then recursively delete the predecessor in the left subtree `root.left = deleteNode(root.left, root.val)`.

- Return `root`.