1. Regarding the structure of AVLTree::remove(AVLNode* &R, int badValue, bool& isShorter) function, let me explain as follows.

   Our intention is to delete badValue (or the node with badValue) from the tree rooted at R (i.e., R is the current node)

    1. If badValue < R’s value → You should make a recursive call remove(R->left, badValue, isShorter) to remove the badValue from the left subtree.

    2. If badValue > R’s value → you should make a recursive call remove(R->right, badVAlue, isShoter) to remove the badValue from the right subtree.

    3. If badValue == R’s value → you need to remove the current node R. For that, there would be 4 different replacement cases. The handling will be explained in the next item.

    After the badValue is removed, you need to update R's bfactor and set isShorter properly, which later will be sent to the R's parent node. Rotation may be needed if a violation is about to happen. All these actions would be all based on the original bfactor & the isShorter flag carried back from the child.

   This whole procedure has a similar logic and structure to the insertion code AVLTree::insertIntoAVL(). The difference is that AVL Insertion does not have the replacement cases.

   Please have a thorough understanding of the insertion coded explained in the lecture notes, including how isTaller is used.

2. For the 4 replacement cases,

    case 1: R is a leaf node ---> go ahead to delete it, set isShorter to TRUE;

    case 2: R is a single-left parent ---> move up the single child, delete R, set isShorter to TRUE;

    case 3: mirror case of case 2;

    case 4: R has two children --> find the predecessor, put the predecessor's value into R; remove the predecessor in the left tree through a recursive call. After the call, check isShorter, update R’s bfactor, and rotate (balanceFromRight) if necessary.

3. In the replacement case 2 and case 3, R has a single child.

In theory, R needs to be removed and the child should be moved up. In practice, however, you don't want to simply remove R, as the connection from the child node to R's parent (the grandparent) will be lost. A better way would be copying the content of the child (everything - value, bfactor, left, right) to R, and then delete the child.