**Algorithm DELETE\_ROOT\_AVL\_TREE()** **Input:** An AVL tree **T** with root node **R**. **Output:** An AVL tree with the root node deleted and the tree rebalanced. **Data Structures:** An AVL tree **T**, node **R**, auxiliary nodes for traversal.

**Steps:**

1. **If R is NULL:**
   1. Return **T** (The tree is empty, nothing to delete).
2. **If R has no children (R is a leaf node):**
   1. Set **R** to NULL.
   2. Return the updated tree **T**.
3. **If R has one child:**
   1. Replace **R** with its child.
   2. Return the updated tree **T**.
4. **If R has two children:**
   1. **Find the in-order successor S of R:**
      1. Set **S** to the minimum value node in the right subtree of **R**.
   2. **Replace the value of R with the value of S:**
      1. Set **R.value** to **S.value**.
   3. **Delete the in-order successor S from the right subtree of R:**
      1. Recursively call **DELETE\_NODE\_AVL\_TREE(T.right, S.value)** to delete **S**.
5. **Rebalance the tree T:**
   1. Update the height of the current node.
   2. **Compute the balance factor BF of the current node:**
      1. **BF = height(T.left) - height(T.right)**
   3. **If the node is unbalanced (BF > 1 or BF < -1), apply rotations:**
      1. **Left Left Case (BF > 1 and BF(T.left) >= 0):**
         1. Right rotate **T**.
      2. **Left Right Case (BF > 1 and BF(T.left) < 0):**
         1. Left rotate **T.left**.
         2. Right rotate **T**.
      3. **Right Right Case (BF < -1 and BF(T.right) <= 0):**
         1. Left rotate **T**.
      4. **Right Left Case (BF < -1 and BF(T.right) > 0):**
         1. Right rotate **T.right**.
         2. Left rotate **T**.
6. **Return the updated tree T.**