AVL TREE

- **Insertion:**

```
AVLNode* insert(AVLNode* node, int key) {
   if (!node) return newNode(key);
   if (key < node->key)
      node->left = insert(node->left, key);
   else if (key > node->key)
      node->right = insert(node->right, key);
   else
      return node;
   node->height = 1 + (height(node->left) > height(node->right) ? height(node->left) : height(node->right));
   int balance = getBalance(node);
   if (balance > 1 && key < node->left->key)
      return rightRotate(node);
   if (balance > 1 && key > node->left->key) {
      node->left = leftRotate(node->left);
      return rightRotate(node);
   }
   if (balance < -1 && key > node->right->key)
      return leftRotate(node);
   if (balance < -1 && key < node->right->key) {
      node->right = rightRotate(node->right);
      return leftRotate(node);
   }
   return node;
}
```

- **Deletion function:**

```
AVLNode* minValueNode(AVLNode* node) {
   AVLNode* current = node;
   while (current->left != NULL)
      current = current->left;
   return current;
}
AVLNode* deleteNode(AVLNode* root, int key) {
   // STEP 1: PERFORM STANDARD BST DELETE
   if (!root) return root;
   if (key < root->key)
      root->left = deleteNode(root->left, key);
   else if (key > root->key)
      root->right = deleteNode(root->right, key);
   else {
      // Node with only one child or no child
      if (!root->left) {
         AVLNode* temp = root->right;
         free(root);
         return temp;
      } else if (!root->right) {
         AVLNode* temp = root->left;
         free(root);
         return temp;
      }
      AVLNode* temp = minValueNode(root->right);
      root->key = temp->key;
      root->right = deleteNode(root->right, temp->key);
   }
root->height = 1 + (height(root->left) > height(root->right) ? height(root->left) : height(root->right));
   int balance = getBalance(root);
   if (balance > 1 && getBalance(root->left) >= 0)
      return rightRotate(root);
   if (balance > 1 && getBalance(root->left) < 0) {
      root->left = leftRotate(root->left);
      return rightRotate(root);
   }
```

```
    if (balance < -1 && getBalance(root->right) <= 0)
        return leftRotate(root);
    if (balance < -1 && getBalance(root->right) > 0) {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }
    return root;
}
```

- **Deletion:**

```
AVLNode* search(AVLNode* root, int key) {
    if (!root || root->key == key)
        return root;
    if (root->key < key)
        return search(root->right, key);
    return search(root->left, key);
}
```