1. private void updateHeight(TreeNode<T> root);

- What it does?
  - This function updates the height of an input node. First, it checks null case when the parameter is null. And it calculate the height of the node. The equation is max(left node height, right node height) + 1. In case of no child on left or right node, it assigns -1 value to those null child.
- Did it use any instance variables of the class, and if so, which ones?
  - No. It only uses the input parameter node and its variable to get a height of a node.
- Are any helper methods called by the method (either of your own creation or in the original codebase), and if so, which ones and what do they do?
  - No. it does not call any helper methods.

2. private int getBalance(TreeNode<T> root);

- What it does?
  - This function gets a balance factor of the input node. First, it checks null case when the parameter is null. If the parameter is null this returns 0. If it is not null, it will calculate a balance factor by left node height minus right node height. This function also treats -1 if child nodes are null.
- Did it use any instance variables of the class, and if so, which ones?
  - No. It only uses the input parameter node and its variable to get a height of a node.
- Are any helper methods called by the method (either of your own creation or in the original codebase), and if so, which ones and what do they do?
  - No. it does not call any helper methods.

3. private TreeNode<T> rebalance(TreeNode<T> root);

- What it does?
  - This function rebalances of the input node. Before starting any rebalance, checks the input parameter is null. And it calls updateHeight() function to update the height of the node properly. And then it figures out what type of rotation is needed for the input node. There are 4 cases here, left, right-left, right, left-right. It determines based of the balance factor of the node and its child node.
- Did it use any instance variables of the class, and if so, which ones?
  - No. It only uses the input parameter node and its variable to rebalance the node.
- Are any helper methods called by the method (either of your own creation or in the original codebase), and if so, which ones and what do they do?
  - Yes. It calls updateHeight(), getBalance(), leftRotate(), rightRotate() functionds. First, this calls updateHeight() before anything to make sure the height of the node is right. And then using getBalance() to find out the balance factor of the node. Once it turns out what type of rotations will be applied, and then calls a proper rotation function.

4. private TreeNode<T> rightRotate(TreeNode<T> root);

- What it does?
  - ■ This function carries out a right rotation of the input node. To make a right rotation, need to define which nodes are going to be related. There are 3 nodes. First, node itself, its parent node, and its left child node. After making rotation, it calls updateHeight() functions twice on the node and its left child node to adjust of the node's height.
- Did it use any instance variables of the class, and if so, which ones?
  - ■ No. It only uses the input parameter node and its variable to make a rotation.
- Are any helper methods called by the method (either of your own creation or in the original codebase), and if so, which ones and what do they do?
  - ■ Yes. It uses the updateHeight() function to set the proper height of the nodes after making rotation.

5. Briefly describe the modifications you made to the insertion and removal functions to maintain height and keep the tree balanced. Where did you initialize the height of a new node?

- Briefly describe the modifications you made to the insertion and removal functions to maintain height.
  - ■ First, set the height of the newly created node inside of the insertHelper function.
  - ■ Second, I positioned rebalance function before returning a root node both of the insertHelper and removeHelper functions.
  - ■ Inside the rebalance function, there is a call to a updateHeight function. Before doing any rebalancing, I have to check whether the node height is correct or not. This is because there are some changes to the node and that means the height must be checked.
- Where did you initialize the height of a new node?
  - ■ I initialize a new node in the case of 'Root is null case' inside of the insertHelper function.
  - ■ After creation of a new node, set the height of the newly created node right away.

For questions 6-8, consider the following proposed extension to our AVLTree class. We'd like to add a method T firstAfter(T v) that, given a value v, returns the least element of the set that is ≥ v. If no such element exists, the method should return a special value "notFound". v itself may or may not be in the set.

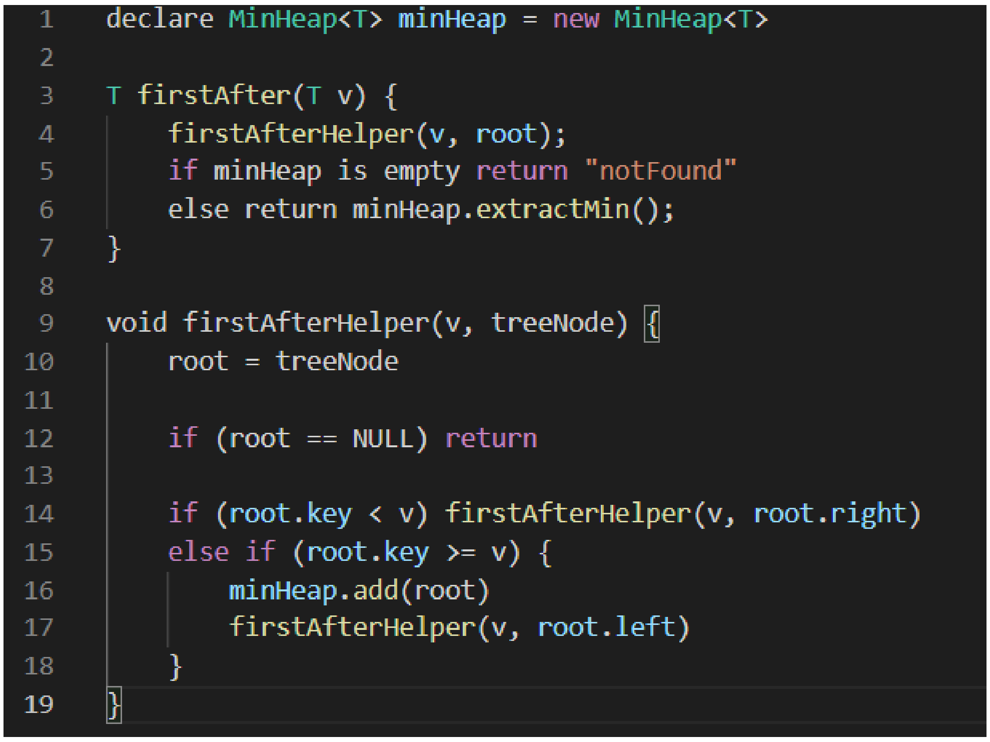
6. Suppose we implement firstAfter() as a top-down tree walk, similar to find(). What should we do if the root of the tree has a key < v? Justify why the behavior you specify is correct.

- If the root of the tree has a key < v, then follow the right subtree to find the node with a value greater than the root.
- Since AVLTree is BST, right subtree is greater than left subtree and the middle node, so we need to find more nodes in the right subtree repeatedly.

7. If the root has a key k ≥ v, what should we do to determine whether k is the least key ≥ v? Justify why the behavior you specify is correct.

- To implement the firstAfter() function, I will exploit a minHeap data structure to store the least element of the set that is ≥ v.
- If the root of the tree has a key ≥ v, then save current node into minheap and follow the left subtree to find the less node than the current node. Repeat this process until it hits the end of the tree.
- Since AVLTree is BST, left subtree is less than middle node or right subtree, so we need to find more nodes in the left subtree repeatedly.

8. Based on your answers above, give pseudocode for an implementation of firstAfter() that runs on a BST in time proportional to its height.



```
declare MinHeap<T> minHeap = new MinHeap<T>

T firstAfter(T v) {
    firstAfterHelper(v, root);
    if minHeap is empty return "notFound"
    else return minHeap.extractMin();
}

void firstAfterHelper(v, treeNode) {
    root = treeNode

    if (root == NULL) return

    if (root.key < v) firstAfterHelper(v, root.right)
    else if (root.key >= v) {
        minHeap.add(root)
        firstAfterHelper(v, root.left)
    }
}
```