Algorithm AVL_Insert(root, key, taller)

Here we assume that each node of the AVL-tree is a structure with 4 components: left_child, key, bf, right_child.

bf can take vales LH (+1), RH (-1), EH (0).

Input: root node of an avl tree, the key data item to be inserted,
and the flag taller indicating whether the tree has become taller

Output: -1 if key is already in the tree, taller = false
otherwise output the root (after the insertion of key) and the
taller flag

Processing: recursively call itself, insert the key at leaf node, and
rotate the tree to maintain balance if necessary

Step1 [Check for base case]
If root is empty, then insert the key at root, taller = true
return.

Step2 [key already in tree?]
If (key = root-> key)
taller = false
return -1

Step3 [recursive left]
If (key < root-> key)
AVL_Insert (root-> left_child, key, taller)
If (taller)
If (root->bf = LH) left_balance(root, taller)
else adjust bf and taller flag
return

Step4 [recursive right]
AVL_Insert (root-> right_child, key, taller)
If (taller)
If (root->bf = RH) right_balance(root, taller)
else adjust bf and taller flag
return