**Pseudocode:**

1. Inserting a node in an AVL Tree:

This pseudocode assumes that the AVL tree has a "node" class with left and right pointers, a "value" attribute, and a "height" attribute. The new node is inserted in the correct position according to the value it holds. The height of the node is updated after the insertion, and the balance factor of the node is calculated. If the balance factor is greater than 1 or less than -1, the tree is rotated to restore balance. The insertion process stops when the node is null, which means that the new node has been inserted into the tree.

PROCEDURE insert(node: Node, value: T)

IF node is null

node = create new Node with value T

return node

ELSE IF value is less than node.value

node.left = insert(node.left, value)

ELSE

node.right = insert(node.right, value)

node.height = max(height(node.left), height(node.right)) + 1

balance = get\_balance(node)

IF balance > 1 AND value < node.left.value

return right\_rotate(node)

ELSE IF balance < -1 AND value > node.right.value

return left\_rotate(node)

ELSE IF balance > 1 AND value > node.left.value

node.left = left\_rotate(node.left)

return right\_rotate(node)

ELSE IF balance < -1 AND value < node.right.value

node.right = right\_rotate(node.right)

return left\_rotate(node)

return node

1. Searching a node in an AVL Tree:

This pseudocode assumes that the AVL tree has a "node" class with left and right pointers, and a "value" attribute. The search starts at the root node, and compares the value of the node with the target value. If the values are equal, the node is returned. If the target value is less than the node value, the search continues in the left subtree. If the target value is greater than the node value, the search continues in the right subtree. If the target value is not found in the tree, the search returns null.

PROCEDURE search(node: Node, value: T)

IF node is null

return null

ELSE IF value is equal to node.value

return node

ELSE IF value is less than node.value

return search(node.left, value)

ELSE

return search(node.right, value)

1. Inorder Traversal (Displaying) of an AVL Tree:

This pseudocode assumes that the AVL tree has a "node" class with left and right pointers, and a "value" attribute. The inorder traversal starts at the root node, and visits the left subtree first, then the root node, and finally the right subtree. The value of each node is printed during the visit. The traversal stops when the node is null, which means that all nodes in the tree have been visited.

In an inorder traversal of an AVL tree, the nodes are visited in **ascending order** of their values. This is because the left subtree of a node always contains nodes with smaller values than the node, and the right subtree always contains nodes with larger values.

PROCEDURE inorder\_traversal(node: Node)

IF node is not null

inorder\_traversal(node.left)

print node.value

inorder\_traversal(node.right)

**Code:**

#include<bits/stdc++.h>

using namespace std;

class Node

{

public:

int key;

Node \*left;

Node \*right;

int height;

};

int height(Node \*N)

{

if (N == NULL)

return 0;

return N->height;

}

int max(int a, int b)

{

return (a > b)? a : b;

}

Node\* newNode(int key)

{

Node\* node = new Node();

node->key = key;

node->left = NULL;

node->right = NULL;

node->height = 1;

return(node);

}

Node \*rightRotate(Node \*y)

{

Node \*x = y->left;

Node \*T2 = x->right;

x->right = y;

y->left = T2;

y->height = max(height(y->left),

height(y->right)) + 1;

x->height = max(height(x->left),

height(x->right)) + 1;

return x;

}

Node \*leftRotate(Node \*x)

{

Node \*y = x->right;

Node \*T2 = y->left;

y->left = x;

x->right = T2;

x->height = max(height(x->left),

height(x->right)) + 1;

y->height = max(height(y->left),

height(y->right)) + 1;

return y;

}

int getBalance(Node \*N)

{

if (N == NULL)

return 0;

return height(N->left) - height(N->right);

}

Node\* insert(Node\* node, int key)

{

if (node == NULL)

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 + max(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->right->key)

return leftRotate(node);

if (balance > 1 && key > node->left->key)

{

node->left = leftRotate(node->left);

return rightRotate(node);

}

if (balance < -1 && key < node->right->key)

{

node->right = rightRotate(node->right);

return leftRotate(node);

}

return node;

}

void InOrder(Node \*root)

{

if(root != NULL)

{

InOrder(root->left);

cout << root->key << " ";

InOrder(root->right);

}

}

int main()

{

Node \*root = NULL;

root = insert(root, 10);

root = insert(root, 20);

root = insert(root, 30);

root = insert(root, 40);

root = insert(root, 50);

root = insert(root, 60);

cout << "Inorder traversal of the constructed AVL tree is \n";

InOrder(root);

return 0;

}

**Output Screenshot:**

****