Day 17 - 12th July 2025

AVL Tree

Adelson-Velsky and Evgenii Landis who invented the AVL Tree in 1962.

AVL trees are self-balancing, which means that the tree height is kept to a minimum so that a very fast runtime is guaranteed for searching, inserting and deleting nodes, with time complexity

O(log n).

The Balance Factor (BF) node (X) = difference in height of Right - Left subtrees.

Bf = 3 - 1 = 2

Balance factor values

0: The node is in balance.

more than 0: The node is "right heavy".

less than 0: The node is "left heavy".

Need clarification on this left - right or right - left

Task 1:

Write Algo for AVL tree

Task 2:

Write code for AVL tree

Hint: try to insert nodes

While inserting get the balance of the tree

Create 2 methods for left rotate and right rotate

Try to insert

Finally display

class AVLNode {

int key, height;

AVLNode left, right;

AVLNode(int key) {

this.key = key;

this.height = 1;

}

}

class AVLTree {

AVLNode root;

// Get height of the node

int height(AVLNode node) {

if (node == null)

return 0;

return node.height;

}

// Get balance factor of node

int getBalance(AVLNode node) {

if (node == null)

return 0;

return height(node.right) - height(node.left);

}

// Update height of the node

void updateHeight(AVLNode node) {

if (node == null)

return;

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

}

// Right rotate

AVLNode rightRotate(AVLNode y) {

AVLNode x = y.left;

AVLNode T2 = x.right;

x.right = y;

y.left = T2;

updateHeight(y);

updateHeight(x);

return x;

}

// Left rotate

AVLNode leftRotate(AVLNode x) {

AVLNode y = x.right;

AVLNode T2 = y.left;

y.left = x;

x.right = T2;

updateHeight(x);

updateHeight(y);

return y;

}

// Insert a node

AVLNode insert(AVLNode node, int key) {

// 1. Perform normal BST insertion

if (node == null)

return new AVLNode(key);

if (key < node.key)

node.left = insert(node.left, key);

else if (key > node.key)

node.right = insert(node.right, key);

else // Duplicate keys not allowed

return node;

// 2. Update height of current node

updateHeight(node);

// 3. Get the balance factor

int balance = getBalance(node);

// 4. Balance the tree if needed

// Left Left Case

if (balance < -1 && key < node.left.key)

return rightRotate(node);

// Right Right Case

if (balance > 1 && key > node.right.key)

return leftRotate(node);

// Left Right Case

if (balance < -1 && key > node.left.key) {

node.left = leftRotate(node.left);

return rightRotate(node);

}

// Right Left Case

if (balance > 1 && key < node.right.key) {

node.right = rightRotate(node.right);

return leftRotate(node);

}

return node;

}

// Search for a key

AVLNode search(AVLNode root, int key) {

// Base case: root is null or key is present at root

if (root == null || root.key == key)

return root;

// Key is greater than root's key

if (key > root.key)

return search(root.right, key);

// Key is smaller than root's key

return search(root.left, key);

}

// Wrapper method for insertion

void insert(int key) {

root = insert(root, key);

}

// Wrapper method for search

boolean search(int key) {

return search(root, key) != null;

}

// Inorder traversal

void inorder(AVLNode node) {

if (node != null) {

inorder(node.left);

System.*out*.print(node.key + " ");

inorder(node.right);

}

}

// Print inorder traversal

void printInorder() {

inorder(root);

System.*out*.println();

}

public static void main(String[] args) {

AVLTree tree = new AVLTree();

// Insert some keys

tree.insert(10);

tree.insert(20);

tree.insert(30);

tree.insert(40);

tree.insert(50);

tree.insert(25);

System.*out*.println("Inorder traversal of the AVL tree:");

tree.printInorder();

// Search for keys

int keyToSearch = 30;

if (tree.search(keyToSearch)) {

System.*out*.println("Key " + keyToSearch + " found in the tree");

} else {

System.*out*.println("Key " + keyToSearch + " not found in the tree");

}

keyToSearch = 60;

if (tree.search(keyToSearch)) {

System.*out*.println("Key " + keyToSearch + " found in the tree");

} else {

System.*out*.println("Key " + keyToSearch + " not found in the tree");

}

}

}

Output:

In order traversal of the AVL tree:

10 20 25 30 40 50

Key 30 found in the tree

Key 60 not found in the tree

11.18 to 11.33 coffee break

Insert operation task 10 min

11.37 to 11.47

Solution 👍

task 1:

Avl insertion Algorithem

1. create a node

2. check if tree is empty or not

3. if tree is empty the inserted node will be the root node.

4. if tree is not empty , do a binary search tree insertion op and also check the balance factor of the node.

5. if the balance factor exceeds 1, we should do rotations on the heavy weighted tree and repeat the insertion from step 4 onwards.

AVL search Algo

1 − Create a node

2 − Check if tree is empty

3 − If tree is empty, new node is root node.

4 − not empty, perform Binary Search Tree insertion operation and check balancing factor of the node in the tree.

5 − Suppose balancing factor > apply rotations on node and resume insertion from Step 4.