**TREE**

Trees: these are hiearical data structures which are used to store and retrieve data efficiently

**TERMS / PROPERTIES OF TREES:**

1. **size:** total number of nodes present in the tree
2. **Child nodes:** a node which is derived from a node above it is called child node
3. **Parent node:**  a node which have a child node I called parent of that child nodes
4. **Siblings :** each parent node can have multiple child nodes and each child is a sibling of other child node of same parent .
5. **Edge :** it is a connection between 2 nodes it is for presentation a relation.
6. **Leaf nodes:** these nodes are ends of tree they does not have further child node or can say they have 0 children
7. **Height:** a height of a node is the maximum number of edges between that node till leaf nodes
8. **Level:** the level is the at what level the node is each time a node have a child it increase 1 level root is at 0 level and with every time we goes to a child node from parent it increase the level by 1 formula : height of root - height of node of which we want the level of
9. **Ancestor :** these are the parent and grand parent node of a node in other words all node which leads to these node
10. **Descendants :** these are the child or grand child node of the parents in other words every node the current node can reach directly or indirectly

**Types of Binary Tree**

1. **Complete Binary Tree :** these are the type of tree where each level are full means contains 2 nodes at each level except for the last level the last level can have any number of node but the node should be filled in left to right way
2. **Full Binary Tree / Strict Binary Tree:** this binary tree each node have either 2 or 0 children
3. **Perfect Binary Tree:** this binary tree each level is full and all leafs node are at the same level
4. **Height Balanced Binary Tree:** here each and every node height difference of left and right subtree is either -1 , 0 , 1
5. **Skewered Binary Tree:**  these tree each node have only a single child (looks like linked list )
6. **Order Binary Tree:** Follow a specific property along each node like Binary Search Tree
7. **Degree Of Tree:** this is the max number of child a node have

**PROPERTIES**

1. **Perfect BST:**  total number of nodes : **2(h-1)+1**
2. **Perfect BST:** total number of leaf nodes**: 2h**
3. If we have n leaves then how many LEVELs we will have

We can have at least log n+1 level

1. **Strict Binary Tree :** if we have n leaf nodes then internal nodes = n-1 not including leaf nodes;

**Traversals**

1. **Pre Order :**

Sequence : (root,left,right)

Used for evaluating Maths Expressions, Serialization ,Copy Tree

1. **In Order :**

Sequence : (left,root,right)

BST inorder traversal give sorted sequence of elements

1. **Post Order :**

Sequence : (left,right,root)

It is used to delete a node from the binary tree

When use bottom up calculation then also we use post order

1. **Breadth First Traversal :**
2. **Depth First Traversal :**

**Disadvantages of Binary Search Trees:**

When we use a BST and all the values comes in a increasing or decreasing order it becomes skewed towards left or right dues to this they becomes like a linkedlist and all operations cost O(n) time   
  
so in order to prevent it we use a self balancing binary Trees to maintain its balance to make the insertion , deletion, searching in Log(n) time

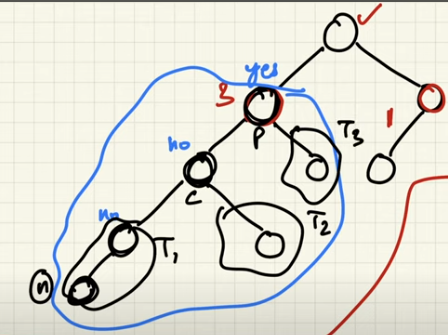
**AVL Tree (Adleson velski and landis)**

**AVL tree is a self balancing binary tree :**

**Algorithm :**

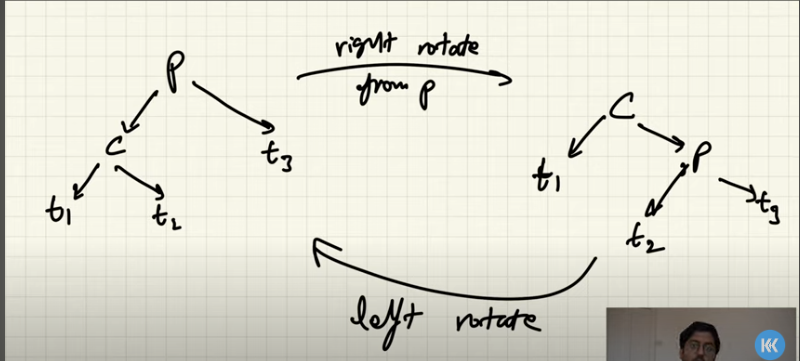
1. **First insert normally insert the value it suppose to be**
2. **Now from inserted Node go up and find the unbalance node**
3. **Use any one of 4 rules and rotate the sub Tree**

**To make is balance we can rotate it to Left Hand Side or right Hand Side**

****

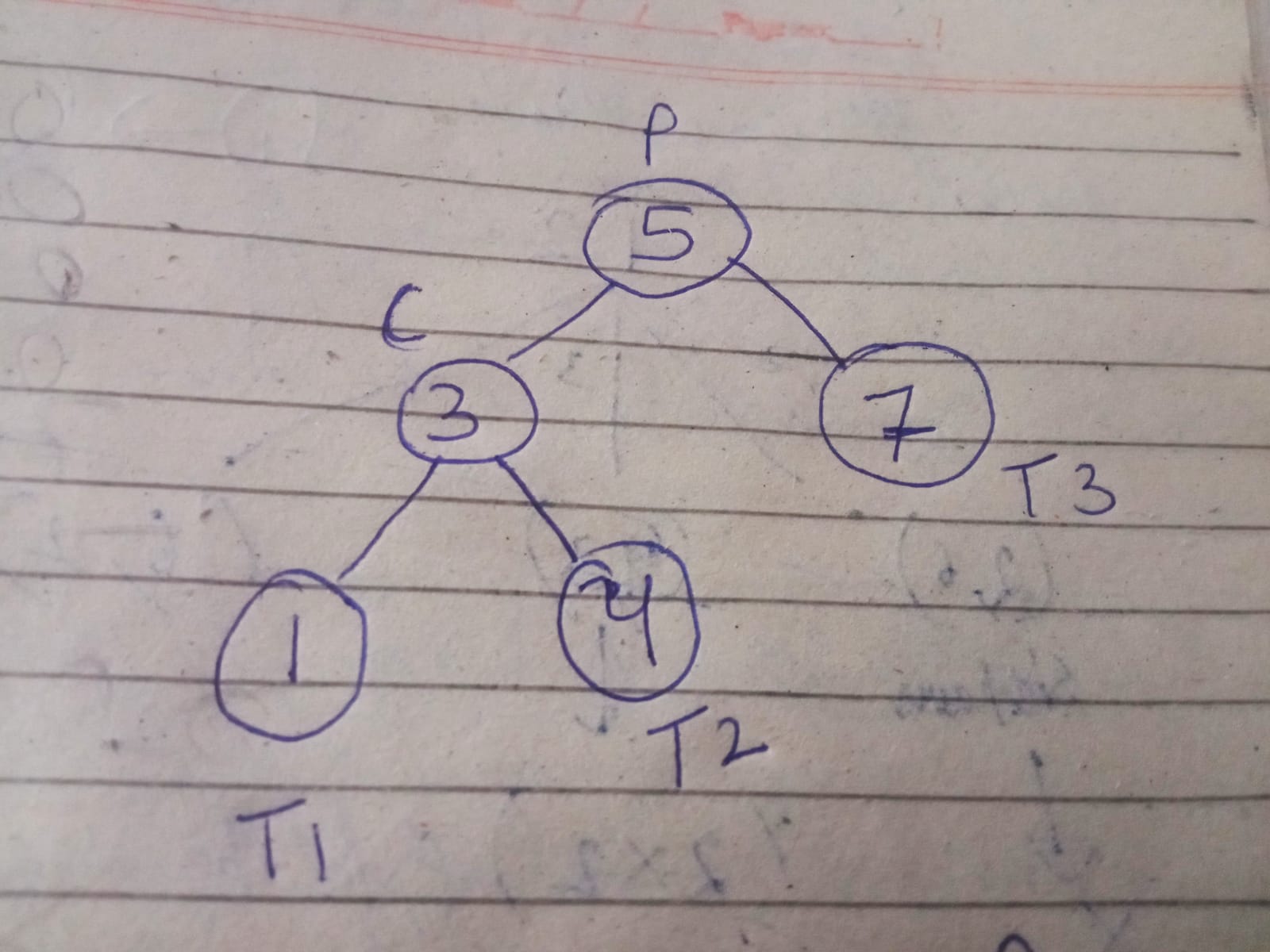
**Rotate Right :**

**Now we can see below**

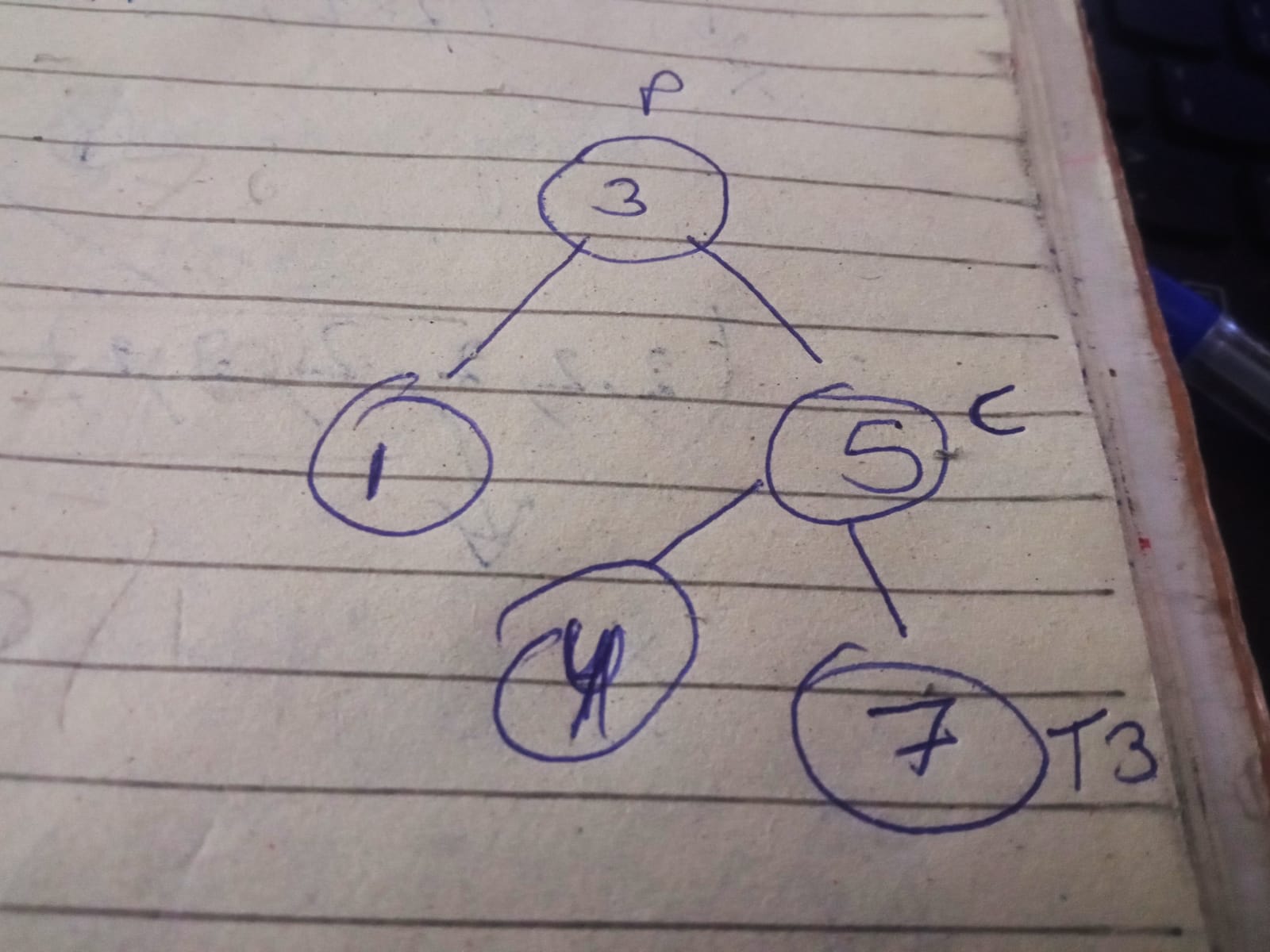


**Example:**

**Before Rotation:**

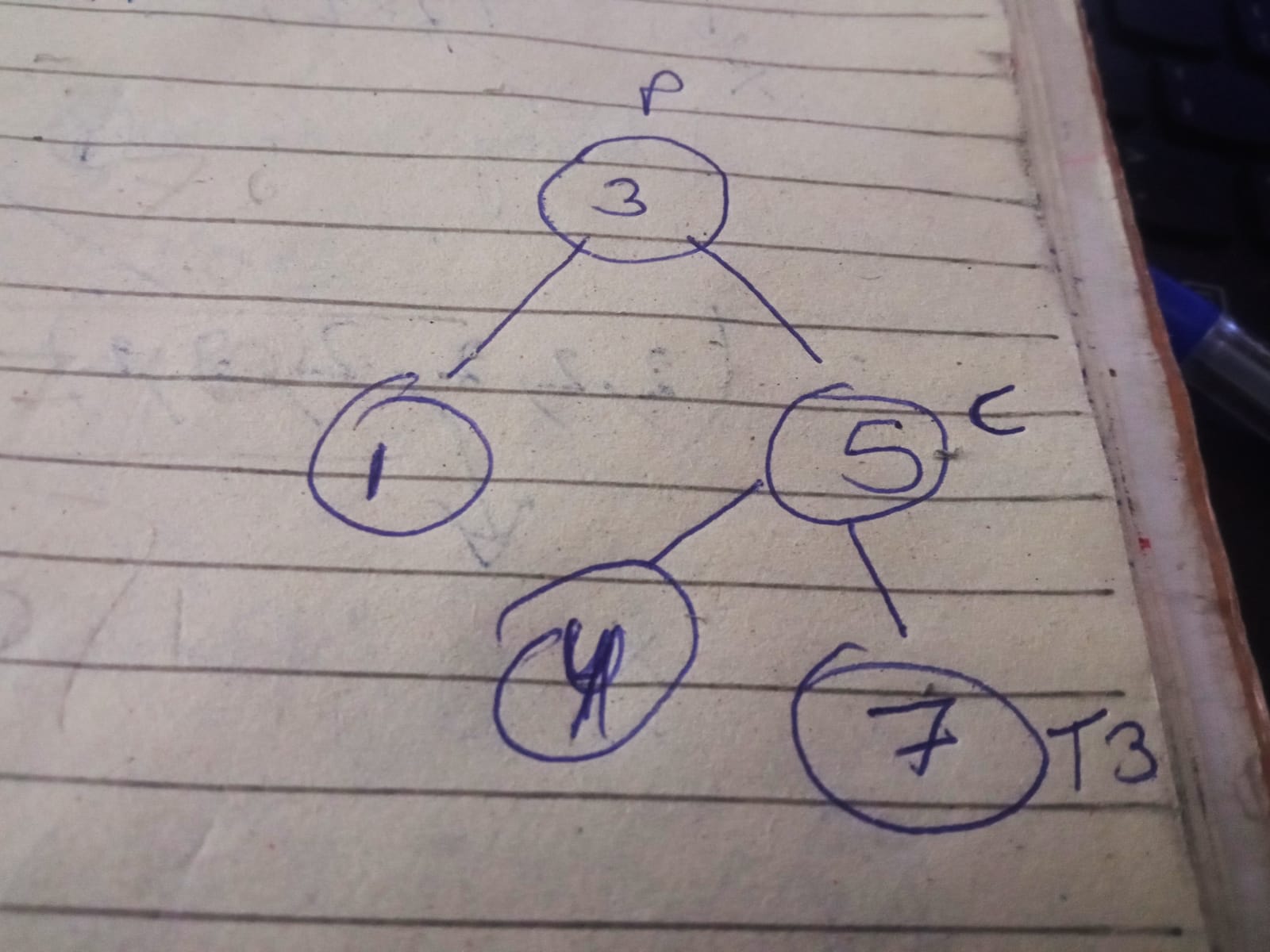
****

**After rotation :**

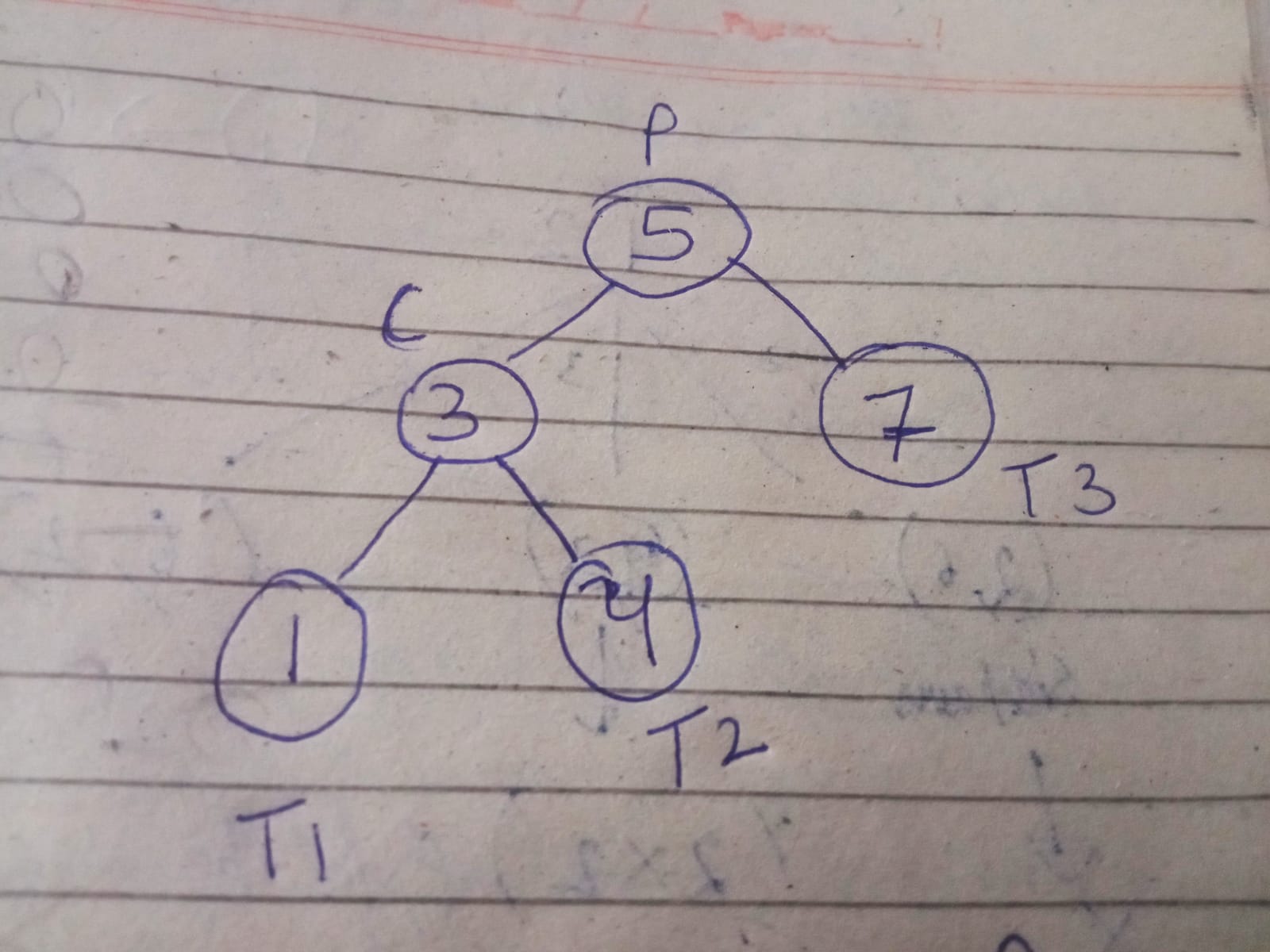
****

**If we want to rotate right its just opposite to what we just did**

**Before Rotation:**

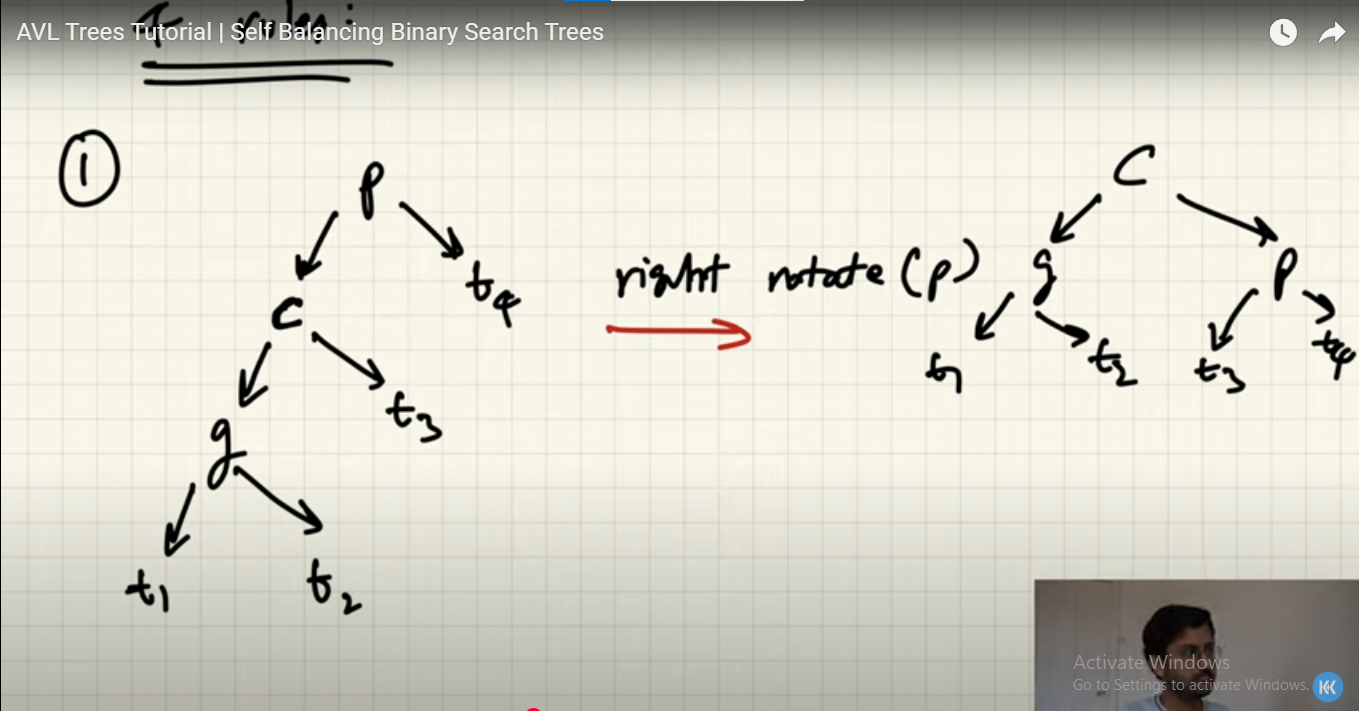
****

**After rotation :**

****

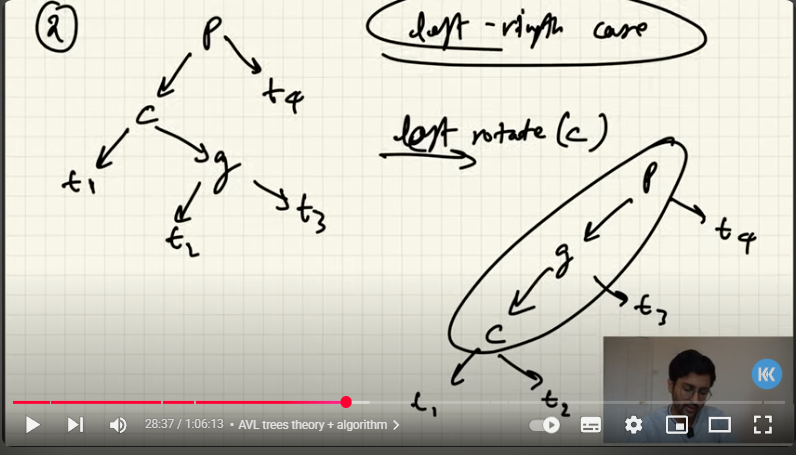
**4 Rules of AVL Tree**

**Rule 1: Left -Left case**

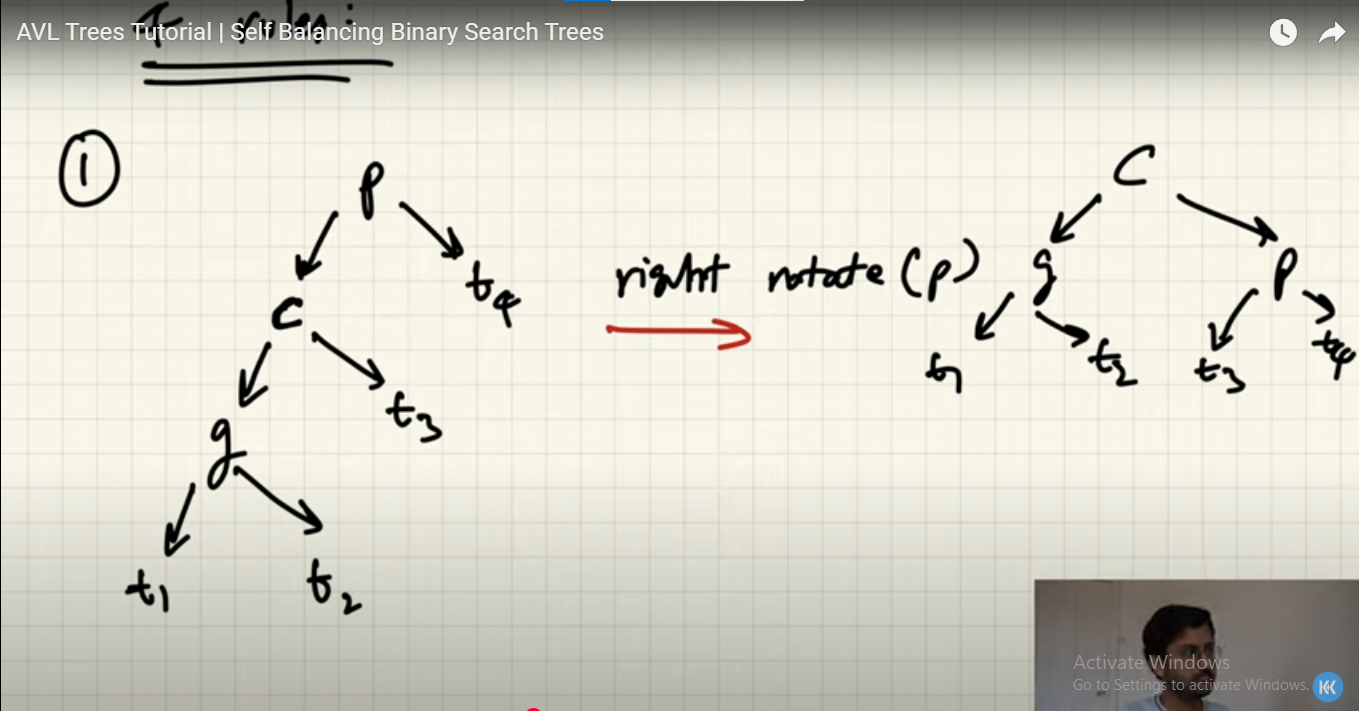


**Rule 2 : Left - Right**

**Here we first Left rotate at C node making it same as Left Left rule**

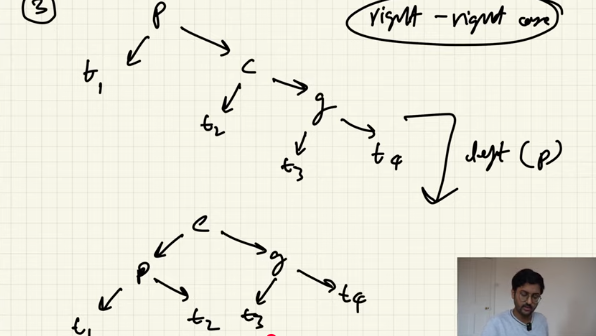


**Then again rotate it to right with parent node same as 1**



**Rule 3 : Right - Right**

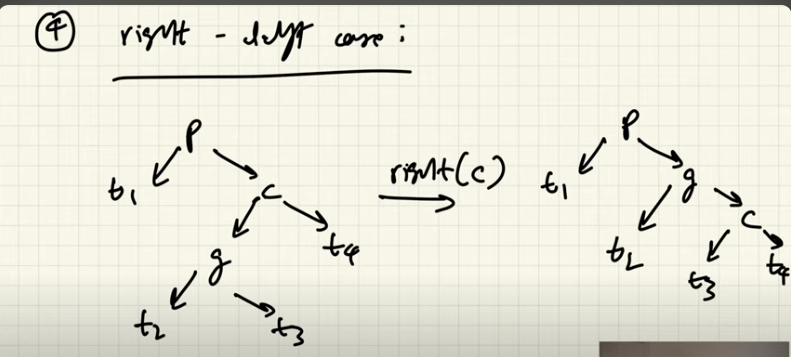
**Here we Rotate just Left one time at P node**



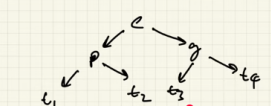
**Rule 4 : Right - Left:**

**First we rotate right form c making is same as Right - Right Rule**

**Then apply left rotation at p to make it balance**



**Left rotation on new p**



**Link for reference : https://www.youtube.com/watch?v=CVA85JuJEn0**