

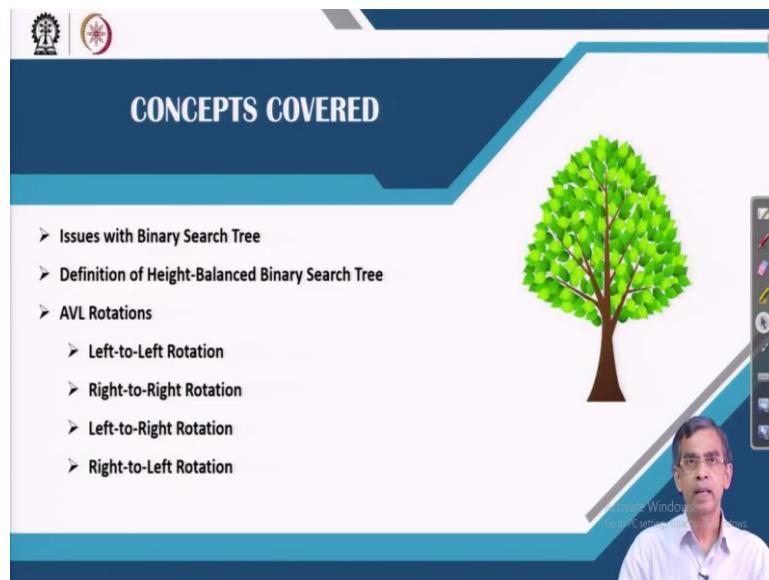
**Data Structures and Algorithms Using Java**  
**Professor Debasis Samanta**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**  
**Lecture: 31**  
**Topic: Height Balanced Binary Search Tree**

So, we have learned Binary search tree which is one important tree data structures. It has many advantages such as searching is very fast. It can store information more efficient manner and then retrieving any elements in it is also very fast. Also it has many applications like sorting a list of elements or searching an elements from a large list.

So, it has many what is called plus points and is a very good candidate data structures to be used in many applications. However, it has one disadvantage as well as. In today's lecture we will learn how a binary search tree can be further improved eliminating the disadvantage that it is having.

So in this direction the many scientist walked for a long and then finally evolved with a new concept. This concept is basically how to maintain a binary search tree which can be created as the best binary search tree and therefore it can sort many applications in a more efficient manner. This concept of new binary search tree is called Height Balanced Binary Search Tree.

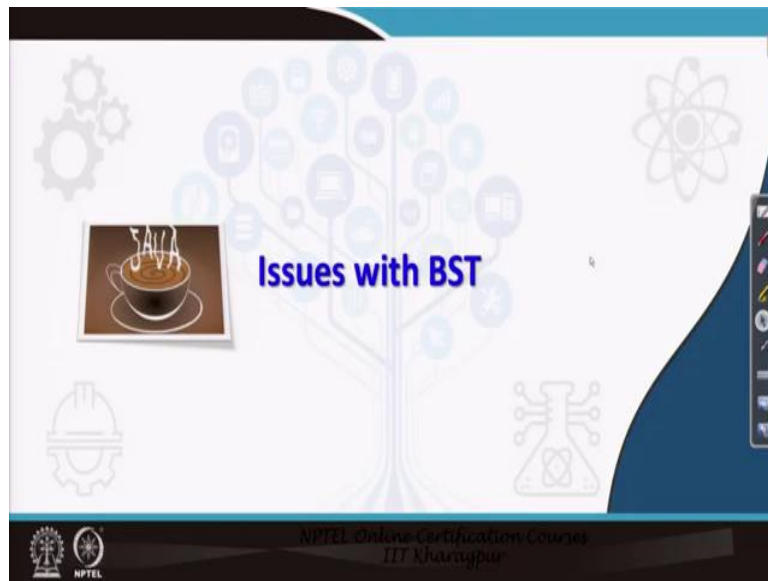
(Refer Slide Time: 2:29)



So, in today's lecture we will try to understand what exactly Height Balanced Binary Search Tree is. That mean when we can make a binary search tree, called Height Balanced Binary Search Tree it is. Actually behind this height balancing a binary search tree there are many operations are required. Those operations mainly called AVL rotation. There are four ways the rotations can be carried out like left to left or right to right, left to right and right to left. And all those rotation if we apply on a

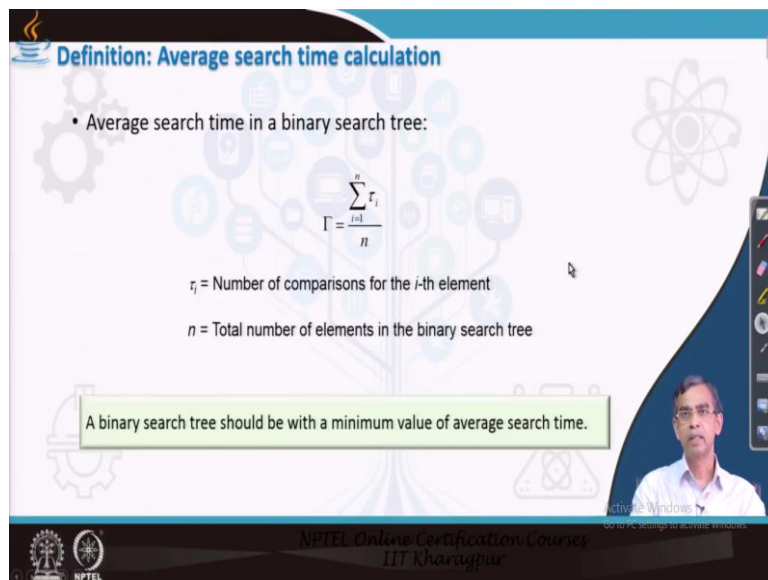
binary search tree, then we can yield a height balanced binary search tree.

(Refer Slide Time: 3:25)



Now, let us come to the discussion of what is the issues with binary search tree and for which we want to have a new binary search tree called height balanced first tree. That means what is the disadvantages that the binary search tree is having.

(Refer Slide Time: 3:43)



Now, I told you that binary search tree has very good application of sorting, searching, traversing, whatever it is there. But the thing is that there are basically if you have given a set of elements then you can construct a large number of binary search tree for the same least in fact. What I want to say, I want to say that for a given list of elements a binary search tree is not necessarily unique. More precisely if same list but in different order and if you build binary search tree with an order then you

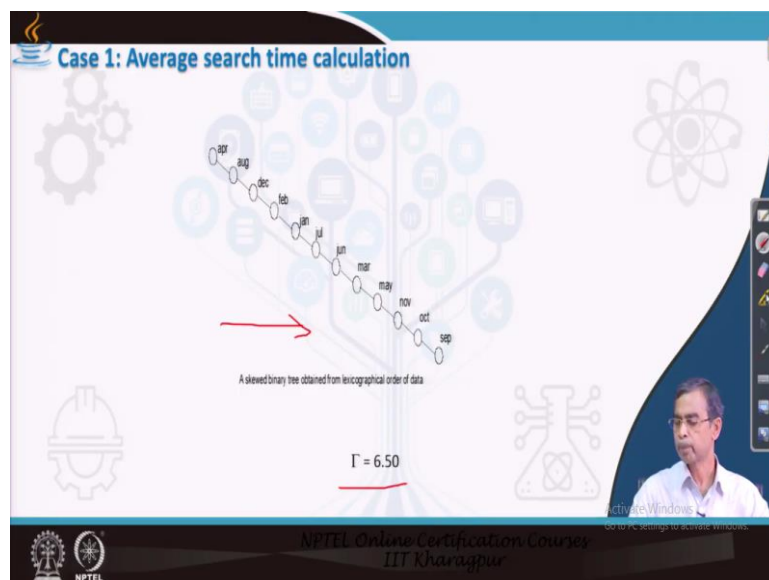
will find another binary search tree.

So, there are different binary search tree is possible even for the same list when the elements in the list are in different orders. Out of so many different binary search trees there is a binary search tree which we can term as optimal binary search tree. Now, what is the idea of optimal binary search tree? A binary search tree is called optimal if we search for an item then every search time it will take minimum.

Now, every search time means we can search an item in a binary search tree and it can be in any place in the tree. Can be in root, can be in the leaf node, can be in the internal node or whatever it is there. So, taking average considering that the target key can be anywhere is called the average search time. I have given a mathematical definition of average search time, it is here. So, in summary what I can say is that binary search tree if we construct it has some average search time value.

And there are many binary search tree is possible even for the same list. Then we have to find an optimum binary search tree for which the average search time is minimum. And this kind of binary search tree is the most efficient binary search tree to be used in applications.

(Refer Slide Time: 6:25)



Now, let us consider an example showing how for the same list the different average search time is possible. Now, here we consider 12 months is a string I can say and suppose these are the list of elements. So, 12 strings are the list of elements. Now, we can have many kind of binary search tree. So, this figure shows one kind of binary search tree and this is called skewed binary search tree.

Now, if we calculate the average search time for this binary search tree using the formula that I have

mentioned in the last slide, then we can say that for this binary search tree the average search time is 6.50. Now, let us consider few more construction of binary search tree for the same number of elements.

(Refer Slide Time: 7:25)

**Case 2 & 3: Average search time calculations**

A binary search tree (half skewed version)  
 $\Gamma = 4.00$

A binary search tree (obtained by inserting the data into the order of months.)  
 $\Gamma = 3.50$

NPTL Online Certification Course  
 IT 60101: Lecture #8  
 Dr. Rajgopal

So, here I have listed few, so this is the one and for this binary search tree the average search time is 4.00, which is better than the previous binary search tree. Now, there is another construction, this is also another binary search tree, but same list of elements and what we can see that average search time is far better, 3.50. Now, let us examine few more binary search tree with same list of elements.

(Refer Slide Time: 7:58)

**Case 4 & 5: Average search time calculations**

Binary search tree in the form of a complete binary tree  
 $\Gamma = 3.08$

A binary search tree obtained from a given random ordering of data.  
 $\Gamma = 3.08$

NPTL Online Certification Course  
 IT 60101: Lecture #8  
 Dr. Rajgopal

Here are few more. And as we can see for this binary search tree is 3.08. As we can see the different way we are constructing and then we are getting better what is called the result. However, this

binary search tree although they are not same, but they both of them have the same average search time. Now, if we study like this building many binary search trees corresponding to a particular ordering of the elements, and then finding the average search tree and then selecting the best binary search tree is, in fact, is a tedious job.

It is sometimes infeasible because if the number of nodes is very large then number of possibilities of binary search tree also increases exponentially and finding and investigating all binary search trees and then finding the best one is not solvable in real time. So, this is an issue. The issue is that finding the best binary search tree in real time is impractical task actually. It is not possible. So, therefore, what is the way?

(Refer Slide Time: 9:13)

The slide is titled "Problem: Finding the best BST". It contains the following text:

- How to find a binary search tree with a minimum value of average search time ?
- Solution
  - Height balanced binary search tree
  - Also called AVL tree (Adelson-Velsky and Landis)
  - Concept of *balance factor* of a node

The formula for the balance factor is given as:  $bf = \text{Height of the left sub-tree } (h_l) - \text{Height of the right sub-tree } (h_r)$

A diagram shows a root node with a left subtree of height  $h_l$  and a right subtree of height  $h_r$ . The slide also features a small video inset of a presenter in the bottom right corner and the NPTEL logo at the bottom left.

The way is that basically we have to have a binary search tree, which basically is a balanced with respect to each node in the tree. Now, what is the balance? Balance means for every node as you know there is left subtree and right subtree. Now, for every tree we know how to calculate height. Likewise the left subtree is also a tree like binary tree as well as right subtree. Now, therefore, for the left subtree we can calculate height and for the right subtree also we can calculate the height of it. Now, if the difference between the two heights of the two subtrees are not so large then we can say the tree is bit balanced.

For example, a skewed binary search tree is heavily imbalanced because for this either left subtree or right subtree height is 0, whereas other is heavy in. So, the difference between the two heights is very large and that is why the average search time taking is also very large. So, in this figure a binary search tree can be made height balanced in such a way that for every node the left subtree and right subtree the height difference is as minimum as possible.

So, in this regard that means given a binary search tree, which is not balanced, how we can make it balanced. We can make it balanced means we can do something on the tree in such a way that, that the heights of any two subtrees having as little difference as possible. So, in this regard, this mechanism to make a binary search tree balanced is proposed by two scientists, mathematician, they are Adelson-Velsky and Landis. They proposed a concept, which basically gives a new form of binary search tree. And it is called AVL tree.

The main idea behind the AVL tree is basically rotation. But before applying this rotation mechanism or rotation procedure, first we have to check whether given a binary search tree is balanced or not. Now, for this purpose they have proposed one term this is called balance factor. Balance factor for each node. Now balance factor for each node is basically the difference between the height of left subtree and right subtrees. So, difference means it absolute value of the difference.

So, it is basically or we can say only difference. If the balance factor of each node is in between minus 1 and plus 1 inclusive 0, then we can say that the binary search tree is balanced. If its value is greater than 1 or less than minus 1, minus 2, minus 3, greater than 1 means 2, 3; then we can say that binary search tree is not balanced. So, now you can apply this balance factor formula for different trees that we have given earlier and we can check that those trees having balance factor greater than minus 1 or less than 1 are having higher average search time actually compared to the others.

So, what we can say calculation of balance factor matters. Now, once we know that the tree is not balanced, then we have to perform rotations. These rotations are called AVL rotations. So, in subsequent few minutes we will try to understand how the AVL rotations are and how those AVL rotations can be programmed. So that given an binary search tree we can make it height balanced.

(Refer Slide Time: 13:55)

Definition of Height-Balanced Binary Search Tree

NPTEL Online Certification Course  
IIT Kharagpur

This slide features a background with a stylized tree and various icons. A small image of a coffee cup is on the left. The title is in blue text. A small video inset of a speaker is in the bottom right corner.

Definition: Height-balanced BST

- Definition

A binary search tree is said to be *height balanced* binary search tree if all its nodes have a balance factor of 1, 0 or -1.

That is, for all nodes  $|bf| = |h_l - h_r| \leq 1$

NPTEL Online Certification Course  
IIT Kharagpur

This slide has a similar background to the first slide. It contains a definition and a mathematical formula. A small video inset of a speaker is in the bottom right corner.

So, let us first try to understand about the definition of height balanced. I have given an introduction that we have to calculate the balanced factor for each node in the binary search tree. And then if we find some nodes or at least any node, at least one node or whatever it is, whose balance factor is greater than, magnitude of the balance factor is greater than 2, greater than or equals to 2, then we can say that tree is not balanced.

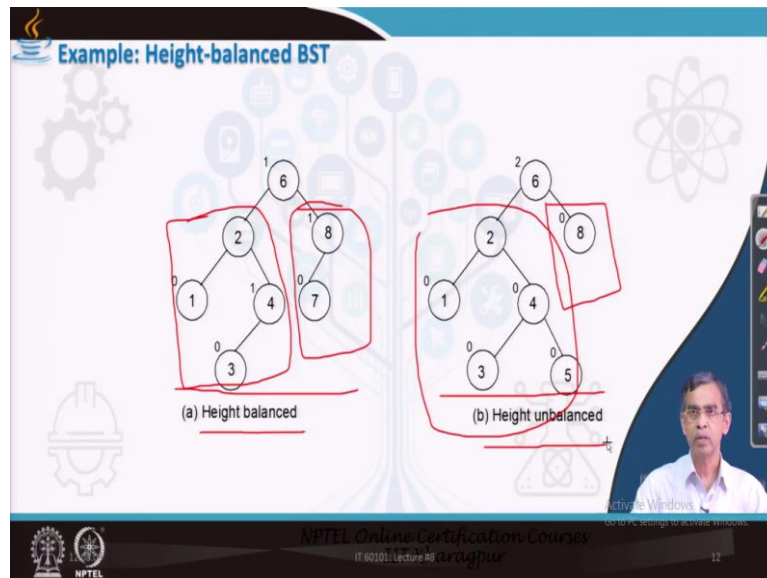
So, here, is a procedure that is required to calculate balance factor for each node in a binary search tree. And in this regard I want to say each time we insert a new node into a binary search tree we have to apply this check that all nodes are having balanced factor in between minus 1 and 1 or not. If it is not, then we have to take action.

That means we have to perform AVL rotation on the tree to make it balance. So, each time we



insert, take a check and then see if it is balanced or not. If it is not balanced perform rotation and continue. So, binary search tree will be built in that way that means in association with the AVL rotations, while we are inserting starting from the empty binary tree. So, finally it will produce the height balanced binary search tree. So, this is basically the working mechanism.

(Refer Slide Time: 15:24)



Now, let us consider few examples to illustrate the concept. So, here I have given one binary search tree, starting from each node if we calculate the height difference then we can find it. For example, for this subtree the height is basically 1, 2, 3 actually. And for this subtree height is 2. So, the difference is 3 minus 2, 1. So, this node has the balance factor 1.

Similarly, for this node also we can check the balance factor, this height is 1, for this node and this is 2, so difference is minus 1. This has the balance factor 0 because it does not have any subtree and so on. So, this way we will be able to calculate the balance factors for each node and then as we see in this case all nodes are having balance factor in between minus 1 and 1, so we can say that this is a height balanced binary search tree. In contrary if we see this is another binary search tree and this is the left subtree and this is the right subtree.

And if we see the balance factor of this node, then we can say that the difference between the heights of the left subtree and right subtree is basically 2. So, this means that it is not balanced. However, all other nodes are having balanced factor in between minus 1 and 1 in this tree. So, therefore, it is height unbalanced. So, at least if one nodes, right, violates the rule then it is not a binary, balanced binary search tree. So, we have understood about height balanced binary search tree.



(Refer Slide Time: 17:20)

**Making a BST height-balanced**

- Steps
  - Insert node into a binary search tree
  - Compute the balance factors
  - Decide the pivot node
  - Balance the unbalance tree

NPTEL Online Certification Course  
IT 60101, Lecture 08 | Arup Kumar

Now, what is the procedure while we are building a binary search tree rather towards the optimum binary search tree, which is basically height balance binary search tree. We have to follow few steps. The first step is that insert the node where it should be inserted using the usual insertion techniques of insertion of a node into binary search tree that we have discussed in the last class. Then once one node is inserted, calculate balance factors for each node, then we have to decide one node which is basically is a criminal node.

Criminal node, in the sense, that it violates the property of a balance binary search tree. That means it has the heights, balance factor which is more than 1. Means it is minus 2 or 2, whatever it is there. Now this is basically is a culprit node and this culprit node maybe termed as pivot node. Again you can recall there may be one or more culprit nodes.

Then again there is a problem is that which culprit we should consider? So, basically the culprit node which is very near to the newly inserted node can be considered as a culprit node. Then we can adjust the balance factor, then again check it if all nodes having the satisfying the balanced factor property then find, if it is not again the next node that is there we have to do one by one like.

But it is observed that only one node basically if it is adjusted, so towards is balanced factor then it become height balance. And all other nodes if it is there whose balance factor is greater than minus 1 or greater than 1 or less than minus 1, automatically their balance factor will be updated to satisfy the property of a height balance tree. So, this is the idea about it. So, detecting the pivot node and once we detect the pivot node we have to balance the tree so balance is by virtue of AVL rotation that we will learn shortly.

(Refer Slide Time: 19:38)

**Making height-balanced BST: AVL rotations**

- AVL Rotations
  - G.M. Adelson-Velskii and E.M. Landis (1962)
- Case 1: Left-to-Left rotation
  - pivot → left-sub tree → left child
- Case 2: Right-to-Right rotation
  - pivot → right-sub tree → right child
- Case 3: Left-to-Right rotation
  - pivot → left-sub tree → right child
- Case 4: Right-to-Left rotation
  - pivot → right-sub tree → left child

NPTEL Online Certification Courses  
IIT Kharagpur

Now, there are many rotations actually, so for a systematic understanding we categorize all the rotation into three cases. These cases are left to left, right to right, left to right and right to left. We will discuss in details how four different cases are possible while we are inserting a node into a binary search tree making a balance search tree become imbalance.

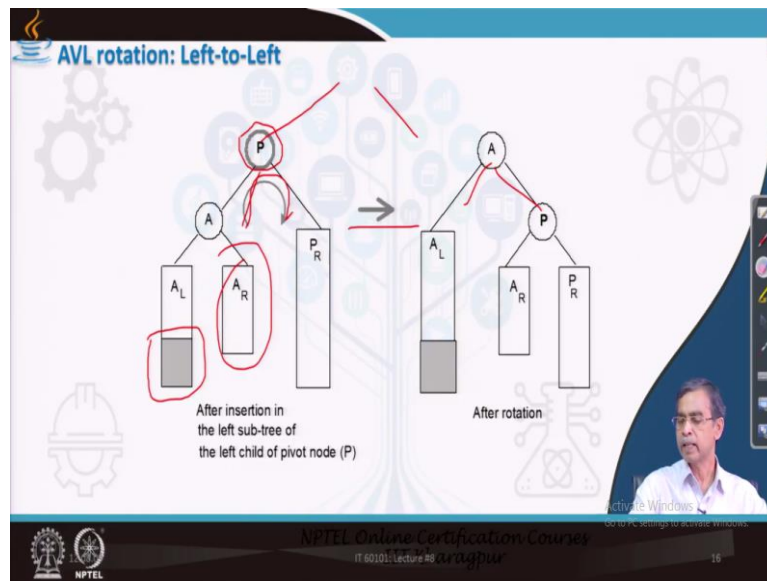
(Refer Slide Time: 20:08)

**Left-to-Left Rotation**

NPTEL Online Certification Courses  
IIT Kharagpur

Now let us first discuss about left to left rotation. Now, the naming left to left or right to right has certain meaning. I will try to clarify what is the meaning of that.

(Refer Slide Time: 20:19)



Now, let us first discuss about the AVL rotation called left to left rotation. Now, let us look at this picture little bit carefully. Now, this is not the entire tree, it may be a part of the tree and there may be another part whatever it is there. Now, we are only focusing to the culprit node that is a pivot node. Let, this is the pivot node whose balance factor violates the property of a balance binary search tree. That means if balance factor is not in between minus 1 and 1, beyond this range.

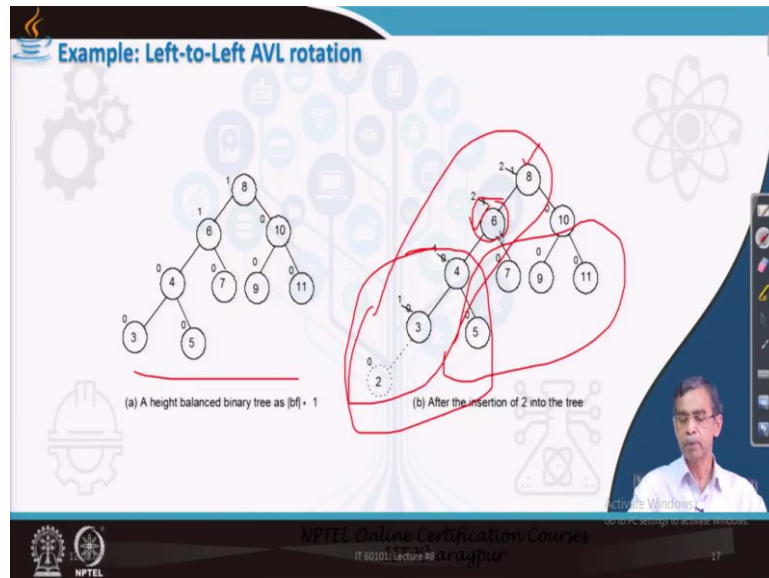
Now, this node become culprit because of a new node insert here. That means a new node is insert into its left subtree and inserted these new node as a left child. So that is why it is called as left to left. That means if a new node is inserted into the left subtree of the pivot node and also it is inserted as a left child, then we may go for left rotation because it basically becomes unbalanced. So, this is the case.

Now, if it is unbalanced then we have to adjust their links. What is the adjustment? Adjustment is that this A should go to P, and P should go down to this place. So, this is the rotation that we have to do. That mean A will go little bit up and P will go down. That means in this case A will take the place of P and P will be the right child of A. If we do this then our left rotation is, left to left rotation is done. Now, here this is the example as we see, this P go down here and A go here so A go there. And this right subtree becomes the left subtree of the pivot node.

And this AR remains the right subtree of the P and A has the only left subtree and the, its right subtree is P actually. So, this kind of rotation is called left to left rotation. I hope you have understood it. If you do not understand then you have to right, thoroughly go through the discussion that I made. And in the book also I have given many applications, many examples rather, you can

go through. And I will go to an example shortly also, so that you can understand it again, so that I will repeat this discussion there. Anyway so idea it is like this.

(Refer Slide Time: 23:05)

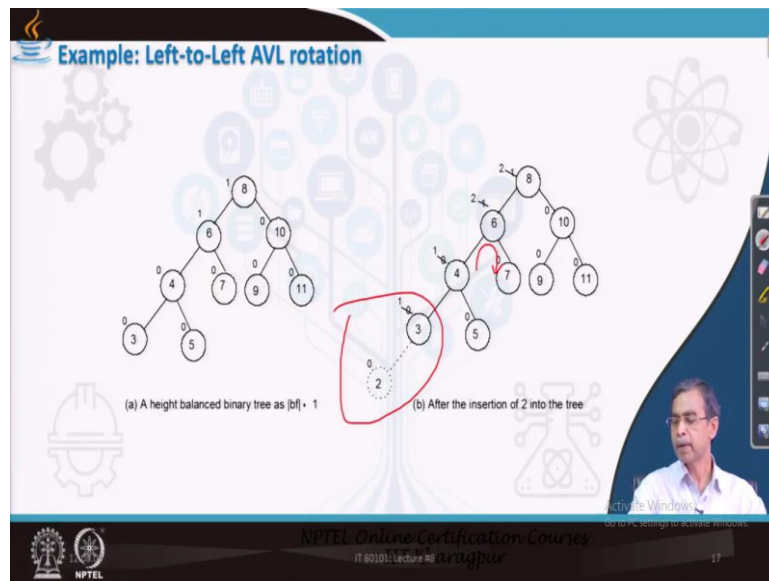


Now, let us now consider an example to illustrate this kind of concept. Let us consider one binary search tree for this purpose. Now, suppose this is the binary search tree, initially it is given and we want to insert a node say 2. So, the location of the point of insertion of 2 in this binary search tree will be here. Now after inserting this node, let us calculate the balance factor of each node.

Now, here you see given initial tree is balanced because this balance factor is within the range minus 1, 2, 1. But when we insert a new node 2, then we can check the change of balance factor where these nodes has not changed the balance factor, but if you see here the nodes change their balance factor.

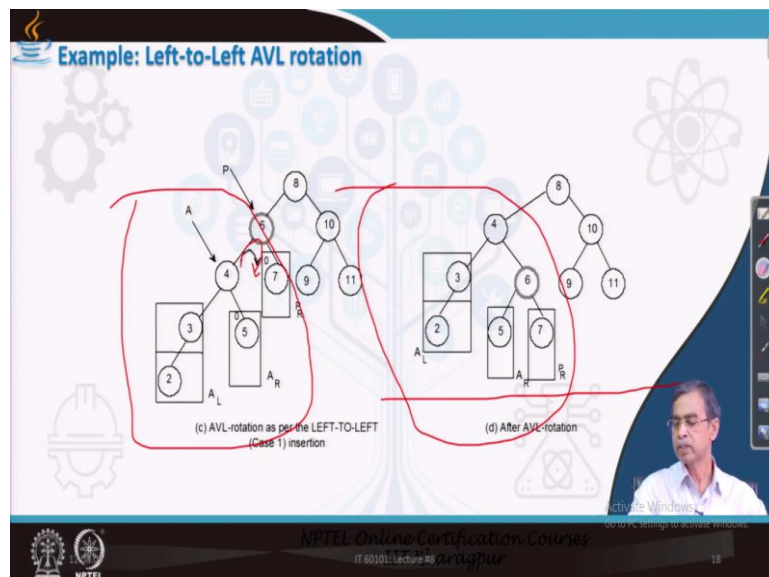
And we see that this node and this node become the culprit node. Out of these culprit node we will consider the nearest node to the point of insertion is the pivot culprit node. So, this node is the pivot node. Now, with respect to this pivot node P, the element is inserted as a left child in the left subtree because it is the left subtree of the pivot node and in this one. So, the rotation that is required in order to make it balance it basically left to left rotation.

(Refer Slide Time: 24:32)



What is the left to rotation? This is the pivot node, so 4, so there is a rotation that we should follow. There is a rotation because it is the pivot node and we have to go for rotation it is like this. This means 6 will come to this place, 7 will be the right child of this. Then left child of 4 will be the left child of the 6 and then 4 will be the place of 6. And it has only the left subtree this one. So, this is the idea.

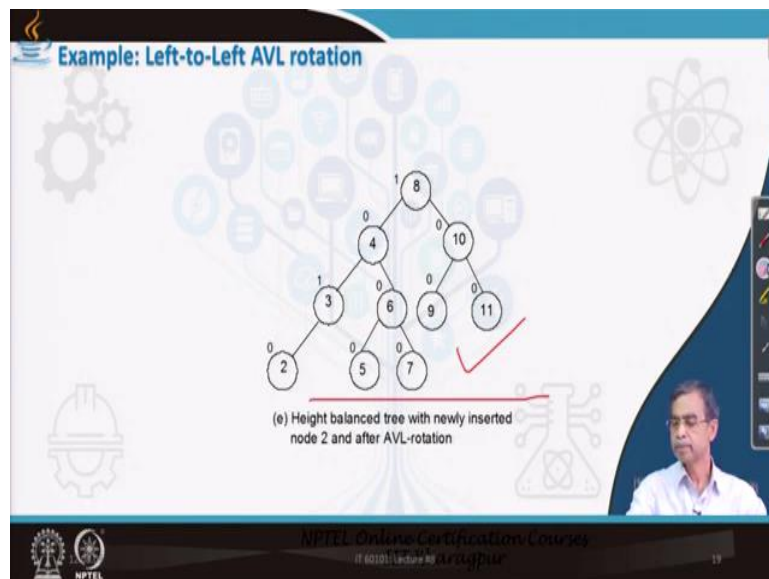
(Refer Slide Time: 25:03)



Now, let us see after balancing that means performing the rotation the new tree how it look like. So, here we can see we make the rotation here. That mean 4 will come to this, 6 will come to this place, 4 will here. So, this part as you see here. So, 4 go to this place, 6 go down and then 5, right subtree become the left subtree of 6. So, it is there and then we can have it. So, now, if we calculate the

balance factor of this node, of this, balance factor of each node in this tree we can see whether it is balanced or not.

(Refer Slide Time: 25:44)

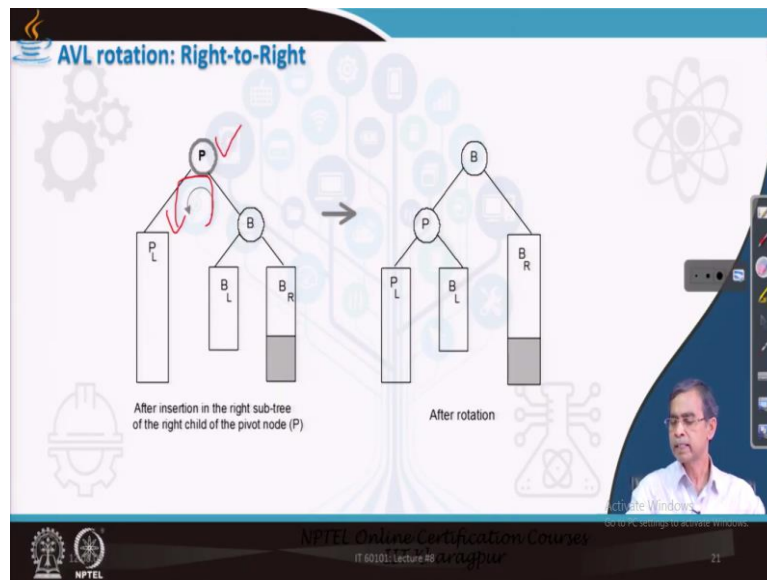


Now, let us see the balance factor of each nodes after this rotation and we can see here no tree violates the properties of the balance. That means balance factor is in between minus 1 and 1, within the range. So, we can say that this is a balanced. And you can note that two nodes are not balanced, in fact, they violates the property but after balancing all nodes satisfy, that is why it is the thing that you need not to go for do many rotation there. So, this is idea about balancing the nodes.

(Refer Slide Time: 26:29)



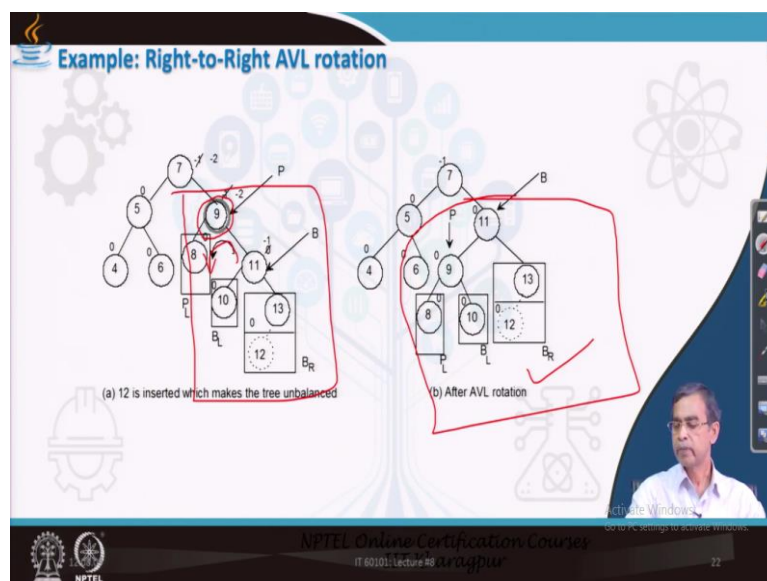




And now let us go for other rotations. So, right to right rotation. Now, left to left rotation we have understood. Right to right rotation is just opposite to the left to rotation. So, this rotation takes place when we insert a node as a right child in a right subtree. Just mirror replica we can say. And then rotation that we should take just opposite so as you say that this rotation we have to take.

It is called right to right rotation. Now, if we do the rotation  $B$  will go to this place,  $P$  will go down and then this part will go to the left subtree of  $P$ . Right subtree of  $P$  and  $B$  becomes this one. So, it is like this. So, you can check it that how the rotation will be performed in order to rearrangement of the trees rather subtrees of different nodes. And here we assume that this  $P$  is a pivot node.

(Refer Slide Time: 27:21)



Now let us illustrate this example with an real binary search tree. Now here is the example. This is the initial search tree given and we wanted to (del) we wanted to insert 12. So, 12 is inserted in the right subtree as a right child or left child it is not a problem, 10 is right. Because here also if you do



no issue. It will do that. For example, if you want to 14<sup>th</sup> also the same thing will happen.

Now, once we insert this node, then we see that balance factor of this node and this node change from minus 1 value to minus 2. So, this will be the pivot node because this is the nearest to the point of insertion. And then with reference to this pivot node we perform this rotation. So, this rotation means 9 will go down as we see that 9 will go down, with respect to this part only. We will not consider this part because that part will remain same. So, this 9 will go down as you see, 11 go up, 11 go up, and then this 10 will go to the right child of 9, so go there and this become. So, this is the tree after the right to right AVL rotation.


Now, after this rotation let us calculate the balance factor of each node and check whether it satisfy the property of height balanced or not. So, here is a balance factor calculation as you can see, in the last, for this node if you calculate the balance factor of each node and we see that no nodes having the balance factor beyond the range minus 1, 2, 1. And therefore, this , this binary search tree is balanced binary search tree.

(Refer Slide Time: 29:12)



**AVL rotation: Left-to-Right**

- Also called double rotation
  - *Rotation 1 (Right-to-Left)*
    - Left sub-tree ( $BL$ ) of the right child ( $B$ ) of the left child of the pivot node ( $P$ ) becomes the right sub-tree of the left child ( $A$ ).
    - Left child ( $A$ ) of the pivot node ( $P$ ) becomes the left child of  $B$ .
  - *Rotation 2 (Left-to-Right)*
    - Right sub-tree ( $BR$ ) of the right child ( $B$ ) of the left child ( $A$ ) of the PIVOT node ( $P$ ) becomes the left sub-tree of  $P$ .
    - $P$  becomes the right child of  $B$ .

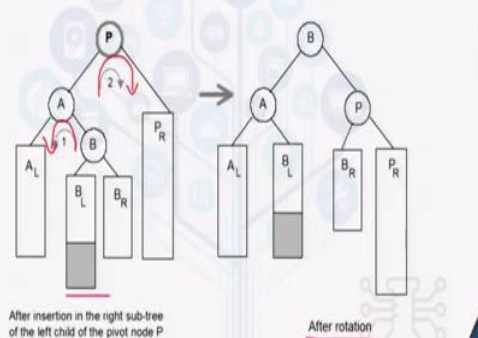


NPTEL Online Certification Courses  
IT 60101: Lecture 48  
Dr. Aravind

Now, let us come to the little bit complex rotation left to right rotation. Now, left to right rotation is basically in contrast to previous two rotation is called double rotation, whereas they are single rotation. So, we need two rotations.


(Refer Slide Time: 29:27)

**AVL rotation: Left-to-Right**



After insertion in the right sub-tree of the left child of the pivot node P

After rotation



NPTEL Online Certification Courses  
IT 60101: Lecture 48  
Dr. Aravind

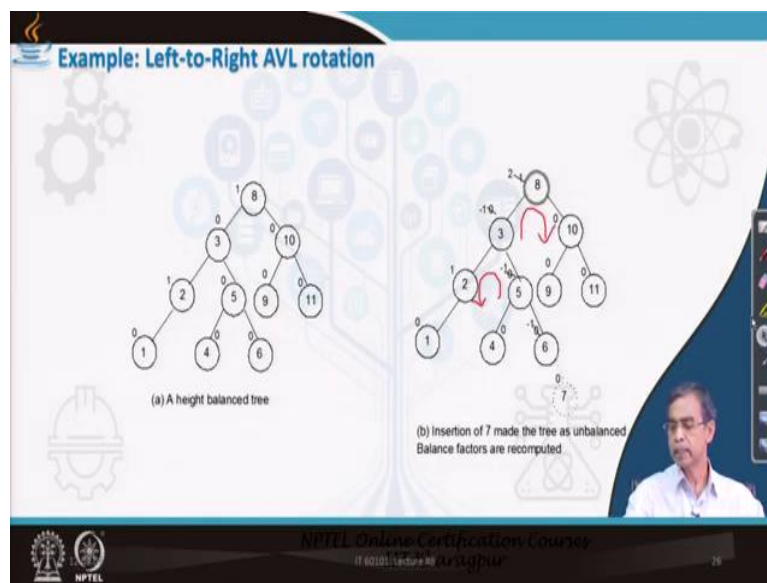
Now, let us see what are the two rotations are there. Likewise, so this is the pivot node and we add the node in the left subtree but as a right, a right child, left subtree, but in the left subtree not in the left. It is left subtree insert in the right part of left subtree. So, if it is like this, so the left to right. Left and then right, so insertion is in the left subtree and then under left subtrees in the right part of the left subtree. So it is like this.

Now if it is like this, then we have to have two rotation. First we do it and then second will do that. So now two rotations should be in order, not that first one and second one. It is not that. This is there where the nearest node is there. So, this is the pivot node, pivot node and next to the pivot

node is this there where the first rotation will be there and the second rotation.

In this rotation if you see it is basically right to right rotation and here if you see left to left rotation. So, two rotations are therefore making the left to right, what is called the rotation, called double rotation and after this rotation the tree, which will look like this it is there, pivot will go there. A will go to the pivot position and here B will, B adjust in such it that BL will go to the right child of A and then B will be the right, P will be the right child of B and this one. So, this is basically before rotation and this is the after rotation.

(Refer Slide Time: 30:58)



Now this is the left to right rotation and here is an example for you. You can understand this so this is the initial tree which is balanced but inserting this one it becomes imbalanced. This you see the left and then left to the right part of the left it is the inserted node. Now you can see the difference between other two single rotation where left, left or left right whatever it is.

Now, here the pivot node, if we calculate it become the pivot node whose balance factor is beyond the range minus 1, 2, 1 and this is also very near to the node where we have inserted currently. And then we have to go the rotation 1, 1 rotation will be this one first, and then next rotation we have to go this one.

(Refer Slide Time: 31:49)

The slide illustrates a Left-to-Right AVL rotation. On the left, a binary search tree is shown with root node 8. Node 8 has a left child 3 and a right child 10. Node 3 has a left child 2 and a right child 5. Node 2 has a left child 1. Node 5 has a left child 4 and a right child 6. Node 10 has a left child 9 and a right child 11. Red arrows indicate the rotation process: node 3 is rotated around node 8, and node 5 is rotated around node 3. Below this tree, it is noted that node 8 becomes the PIVOT node. On the right, the resulting balanced tree is shown. The root is node 5, with left child 3 and right child 8. Node 3 has left child 2 and right child 4. Node 2 has left child 1. Node 8 has left child 6 and right child 10. Node 6 has left child 9 and right child 11. Below this tree, it is noted that the tree is balanced after the rotation. The slide includes the NPTEL logo and the text 'NPTEL Online Certification Courses IIT Kharagpur'.


Now here is the example that we can check how this two rotation if we perform and here I show it. Now, here the two rotations it is shown here. This is the first rotation and this is second rotation and after the rotation adjustment the binary search tree will takes place. And if we calculate the balance factors of all nodes in this tree, which is that they are within the range. Therefore, it is after rotation it becomes balanced. So, this is the procedure about left to right AVL rotation.

(Refer Slide Time: 32:14)

The slide is titled 'Right-to-Left Rotation'. It features a central graphic of a coffee cup with steam rising from it. The background is a light blue grid with various icons like gears, a tree, and a molecular structure. The NPTEL logo and the text 'NPTEL Online Certification Courses IIT Kharagpur' are visible at the bottom. A small video inset of a speaker is in the bottom right corner.

**AVL rotation: Right-to-Left**

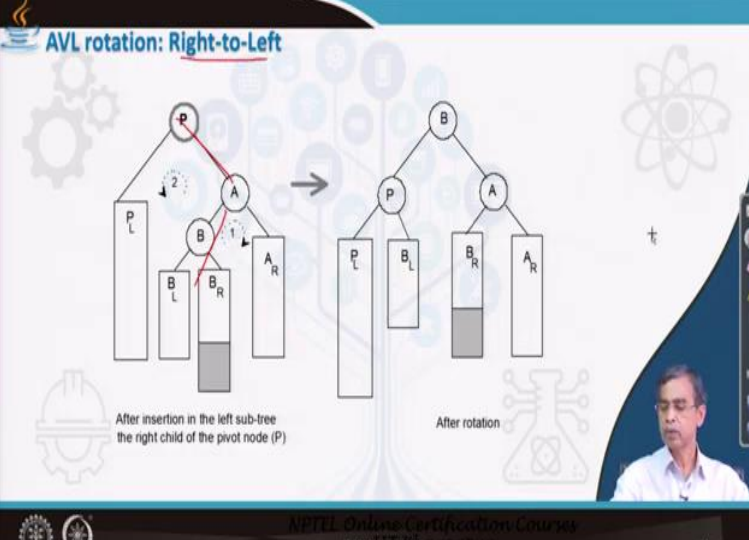
- It is also another double rotation
  - *Rotation 1 (Left-to-Right)*
    - Right sub-tree (BR) of the left child (B) of the right child (A) of the pivot node (P) becomes the left sub-tree of A.
    - Right child (A) of the pivot node (P) becomes the right child of B.
  - *Rotation 2 (Right-to-Left)*
    - Left sub-tree (BL) of the right child (B) of the right child (A) of the pivot node (P) becomes the right sub-tree of P.
    - P becomes the left child of B.



Now likewise right to left AVL rotation is very similar to the previous rotation.


(Refer Slide Time: 32:18)

**AVL rotation: Right-to-Left**



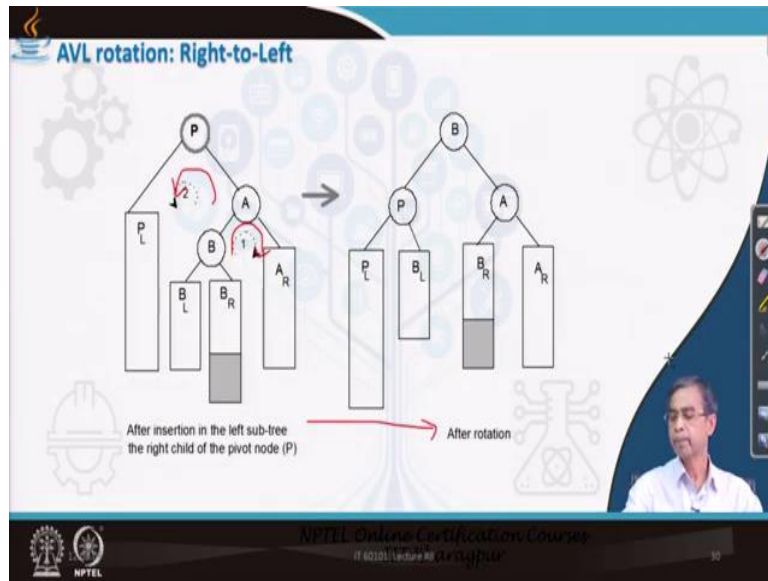
After insertion in the left sub-tree the right child of the pivot node (P)

After rotation



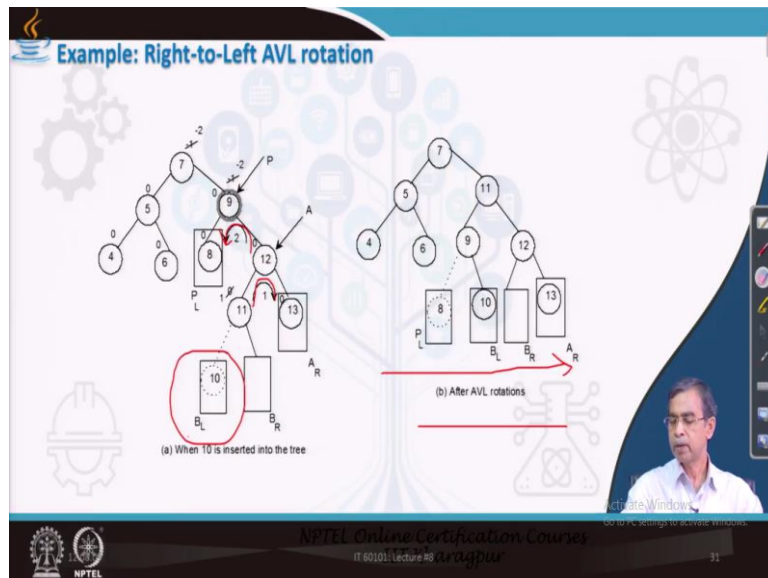
Here, also two rotations are there, but here in the case that it should, I mean the tree become unbalanced because of inserting a node into a right subtree of the pivot node and again right child of this one. So, this is left child of this one. So, it is called right and then it is added in the some part of the left subtree at the right of the pivot node. So, that is why it is called right to left.

(Refer Slide Time: 32:48)



Now, here the rotation that you should perform, two rotations right. So, near to the insertion point this is the first rotation and as we see left to left and then this is right to right. And here this rotation is the left. This is okay, this is left to left and this is right to right rotation. So, two, I mean single rotations compute the double rotations there and after the rotation this become the tree it is there. This is the double rotation.

(Refer Slide Time: 33:18)



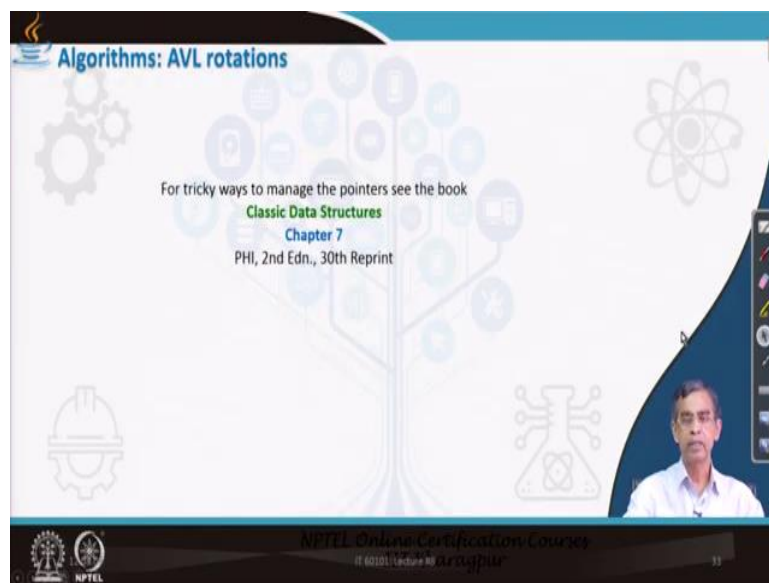
Now, let us have one example to understand this concept more clearly and here this is the input binary search tree and we insert the node here. So, it is basically pivot node become, this is the pivot node which is nearest to this one. And nearest to this means it is basically node which is newly inserted in the right subtree as a left part of right subtree, its right subtree.



So, there are two rotations; one is this one called left to left and this is another right to right rotation, which basically 9 will go down, 12 will go to the place of 9 and this part will go to the child of 9. And then the rotation is done, so it is like this. So, 12 and this one it is there. Now, this is basically the tree after balancing and we can calculate the balance factor of this node and we will see that it becomes balanced.

So, this is the idea about 4 different rotation. They are called AVL rotation according to the name of the scientist who first invented the Adelson-Velksy and Landis. So, this type of rotation.

(Refer Slide Time: 34:23)



Now, we have learned about the different rotation and then we can understand about how the pointer manipulation is there. Now regarding pointer manipulation details algorithms are there. I advise you to go to this book to study the different algorithms there for corresponding to different rotations.



(Refer Slide Time: 34:38)

The slide is titled "Height of an AVL tree" and features a background with a stylized tree and various icons. It contains the following text:

- Maximum height possible in an AVL tree with  $n$  number of nodes is given by

$$h_{\max} = 1.44 \log_2 n$$

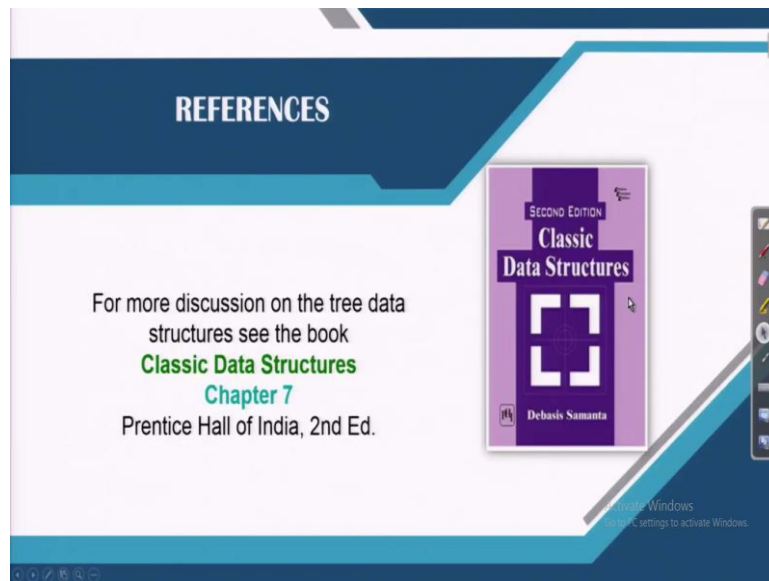
For the proof of this property see the book  
**Classic Data Structures**  
Chapter 7  
PHI, 2nd Edn., 30th Reprint

At the bottom of the slide, there is a footer with the NPTEL logo and the text "NPTEL Online Certification Courses" and "IT 60101: Lecture 09 - AVL tree".

Now after performing the height balancing of binary search tree, we will be able to see that average search time of this tree is minimum compared to any other tree in fact. Therefore, finding the best tree is possible or feasible if we do the balancing after insertion of each node starting from the empty node. And it is also observed that height of balance factor is always minimum and the maximum height that is possible for a height balance tree is this point. So, this is the minimum height possible and this is maximum value that is the minimum height is possible.

So, no binary search tree with  $n$  number of nodes can go beyond this height actually. So, this is the maximum height is possible and this is the height that is possible for a minimum binary search tree or balance binary search tree. And the details calculation, discussion you can find in this book, in chapter 7 also. You are advised to check it how we have prove this formula.

(Refer Slide Time: 35:43)



Now for further study definitely should ask you to go through this book. To study much more and then programming and everything obviously it is not so easy, but tedious. But if you follow algorithm and try to understand the algorithm and then implement using programming that will not be a big job for you.

Thank you very much and hope for a better learning and then I should advice you to study the same things several times because it is little bit complex topic rather. It is advance topics, then many people at the first reading can find it difficult to understand. B11ut I hope that you will be able to follow either from my lecture as well as form the book where I have referred. Thank you very much.