Data Structures and Algorithms

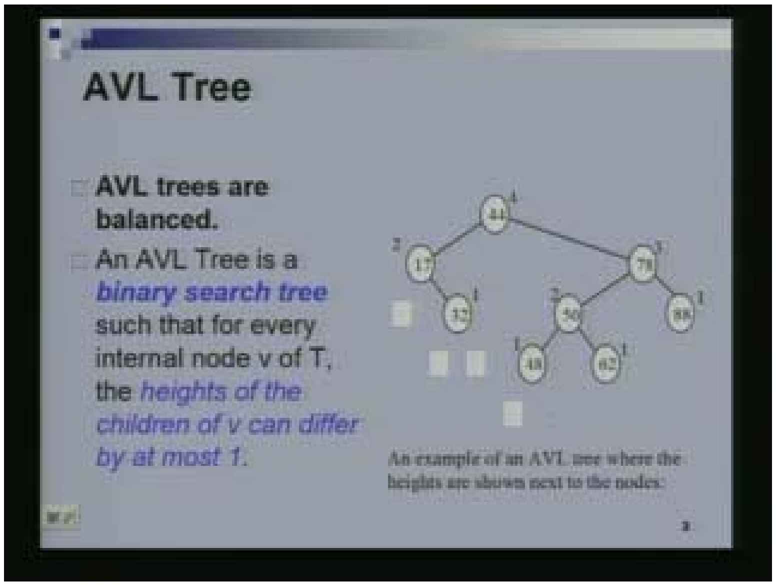
Dr. Naveen Garg
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

Lecture – 11

Avl trees

In this class we are going to talk about AVL trees. In the last class we have seen binary search tree data structure. One problem with the binary search tree if you recall is that the operations of insertion, deletion and search take time proportional to the height of the tree. Height of the tree can be very bad. We saw an example were the height of the tree could be as bad as order n or n-1. We want to some how create a tree which does not have too bad a height. That is what we are going to do today. We are going to look at this data structure called AVL trees. What is an AVL tree? AVL trees are also called height balanced trees. Ignore the white spots that are showed on the slide below and they should not have shown here.

(Refer Slide Time: 01:58)



AVL Tree

- **AVL trees are balanced.**
- An AVL Tree is a *binary search tree* such that for every internal node v of T, the *heights of the children of v can differ by at most 1.*

An example of an AVL tree where the heights are shown next to the nodes:

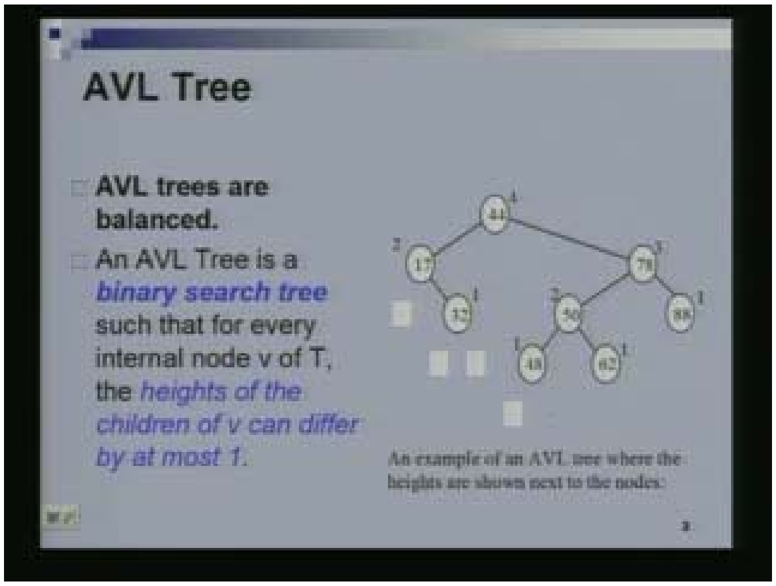
3

This is the binary search tree and inside the nodes are the keys. Everything which is less than the root is to the left of the root and everything which is more than the root is to the right of it. The thing that is written next to each node is the height of a node. What we will call the height of a node? We have not defined this term yet. We will just say the height of a node is the height of the sub tree rooted at that node.

For instance if I look at this node (78), all the things which is below it is the sub tree rooted at this node. What is the sub tree rooted at a node? It is just the set of descendents. I am looking at the tree which is on the right and in previous classes we have defined the height of such a tree as 2 and not 3. Because we had said that 78 is at level zero and 50 is at level 1 and 48 is at level 2 and so we called the height of the tree as 2. We will just modify, we will say that if it is the singleton node just one node then it is of height one instead of height zero as we have been calling it. So level numbers are beginning with 1. This sub tree 50 has height 2 and this sub tree 78 has height 3 and this entire tree has height 4. We are going to call this as height of the tree for the purpose of the AVL tree.

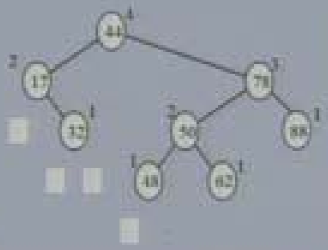
With every node I have put down the height of that node. What is the height of the node? It is just the height of the sub tree rooted at that node. All the leaves will have height 1, the parents of the leaves will have height 2 and so on. Such a tree is called AVL tree if it is height balanced. What is height balanced? If I look at any node and its children then the difference in their height is at most one. There might be no difference in their heights, as in the case with the 50th node. Its 2 children have the same height. The node 78 has the difference, the left sub tree has more height than the right sub tree. The left sub tree has height 2 and the right sub tree has height 1. The node 44 also has a difference of one. The right sub tree has height 3 and the left sub tree has height 2. But the difference is no more than a one. This is the AVL tree. This is what our definition of an AVL tree would be. It is true for every node of the tree. The binary search tree has 2 properties. It has to be a binary search tree and for every internal node of the tree, the heights of the children differ by at most one. Why have I said internal? For a leaf node it has no children. It does not make any difference to talk about the height of the tree. So for this node 17 the right sub tree has height one. The left sub tree is missing so we call it height zero. Now you understand why I had made this change. If the tree is absent then I will denote the height as zero. And the single node will become height one. That is why I have to shift the definition a little bit.

(Refer Slide Time: 6:04)



AVL Tree

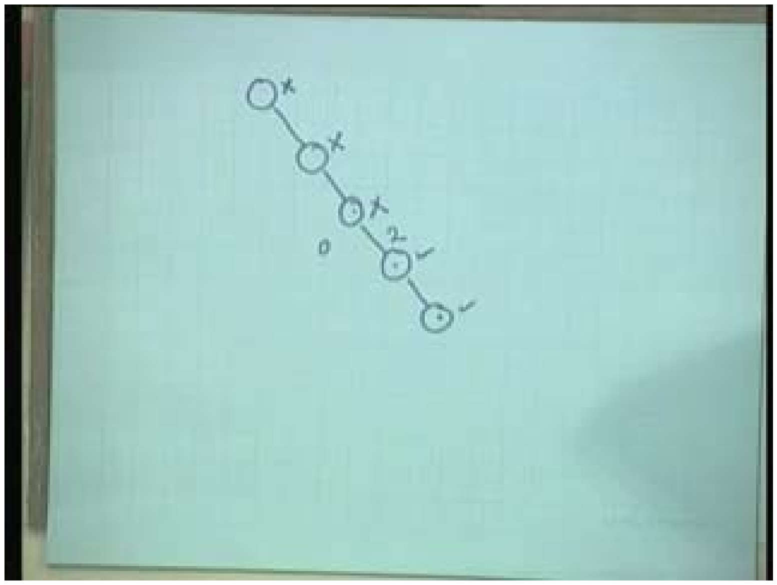
- **AVL trees are balanced.**
- An AVL Tree is a ***binary search tree*** such that for every internal node v of T, the *heights of the children of v can differ by at most 1.*



An example of an AVL tree where the heights are shown next to the nodes:

Let us see what is not an AVL tree? So recall that one of our binary tree which was very bad, which had a huge height was a tree like this. This is a binary search tree and I put some keys so that it looks like a binary search tree. This has height equal to n-1, if there were n nodes. Is this an AVL tree? No. Is the last node height balanced? Yes, since it is a leaf node it is height balanced. Is the next node height balanced? Yes, it is also height balanced. Is the node following the 2nd node height balanced? No because the right sub tree has height 2 and the left sub tree has height zero. Thus the height balanced property is violated here. It is also violated in the following nodes. Thus we will never have such kind of trees as AVL trees.

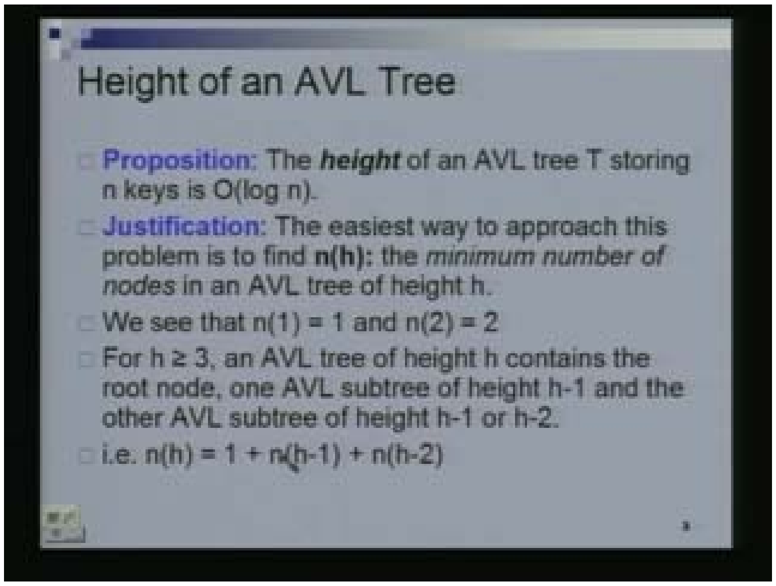
(Refer Slide Time: 07:22)



Since we said that we are not going to have such a kind of trees as AVL trees, let us try and figure out how bad the height of an AVL trees can be. Let say I have an AVL tree of n nodes, if its height can still be as bad as n-1 I have not gained anything. I would like to say that its height is no more than log n or something. We will figure that out and that is what we are going to prove in the next few minutes. The height of an AVL tree t which has n nodes in it is only order log n. Let see why this is true. I am not going to prove this claim directly, I am going to make a slightly different argument. Let us take an AVL tree of height h. Amongst all possible AVL trees of height h, let me see the one which has the smallest number of nodes. I defined this quantity n (h) as the minimum number of nodes in an AVL tree of height h. Let us figure out the quantity and then we will see how this implies the proposition.

Given an AVL tree of height h, we want to find out what is the smallest number of nodes it has. Can it have only h nodes? Then we will be in trouble. We want to say it has many nodes, if you recall a binary search tree of height h can have only h+1 nodes like the example that I showed you. But a good tree which is like a complete binary tree of height h will have 2^h nodes. What we would really like is that our AVL tree which was of height h has large number of nodes, not just h but more like 2^h or something like that. That is what we are going to prove.

(Refer Slide Time: 11:31)



Height of an AVL Tree

- **Proposition:** The ***height*** of an AVL tree T storing n keys is O(log n).
- **Justification:** The easiest way to approach this problem is to find **n(h):** the *minimum number of nodes* in an AVL tree of height h.
- We see that n(1) = 1 and n(2) = 2
- For h ≥ 3, an AVL tree of height h contains the root node, one AVL subtree of height h-1 and the other AVL subtree of height h-1 or h-2.
- i.e. n(h) = 1 + n(h-1) + n(h-2)

Let us understand the quantity. It is the minimum number of nodes in an AVL tree of height h. What is an AVL tree of height 1? It is just a singleton node and nothing else. It has only 1 node in it. If I have an AVL tree of height 2 then it has root and 1 node. But it can also be root and 2 children. Why have I written n (2) = 2 and not 3 because I am counting the minimum. That is why n (2) =2, the minimum number of nodes will be just 2 in an AVL tree of height 2.

Suppose if I have an AVL tree of height 3 or more, it will contain 1 root node. Suppose if I have an AVL tree of height h, it will contain 1 root node and an AVL tree of height h-1 on one side and an AVL tree of height h-2 on the other side. Why h-1 and h-2? It has height h so its children can have height only h-1 and not more than h-1. They can have a difference of at most one. If one of them is h-1 the other one can only be h-2 or h-1. One of the sub tree has height h-1 and the other sub tree has height h-1 or h-2. But what will we pick? We would like that the other sub tree should have height h-2. Why? Because of minimum number of nodes. A tree which has smaller height will also have smaller number of nodes, so we would like that the height of the other sub tree to be h-2.

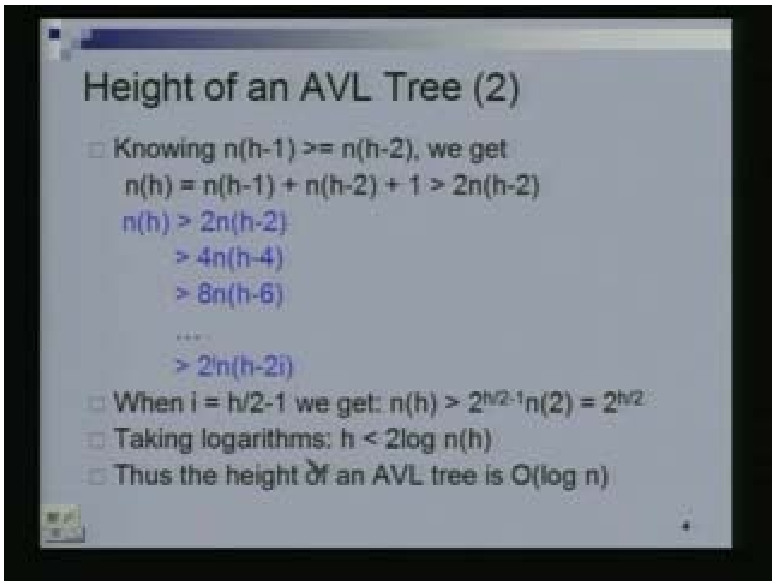
If n (h) was the number of nodes in the tree of height h, then what is the number of nodes n (h) equal to? It is the number of nodes in a tree of height h-1 the smallest possible, because the left sub tree which is of height h-1 can have as small as little number of nodes as possible and in the right sub tree which is of height h-2 also has little number of nodes as possible. The number of nodes in the left sub tree is n (h-1), the number of nodes in the right sub tree is n (h-2). There is one root node and the recurrence relationship would look like this (n (h) = 1+n (h-1) + n (h-2)). Once again we are seeing the recurrence relation. This is what we have to solve today. What are the base conditions? We know n (1) is 1 and n (2) is 2. With that you can figure out what n (3) would be? n (3) would be 1+1+2 which is 4 and so on. But we would like a close form expression to do this. So we will solve this recurrence.

We are not going to be solving this recurrence exactly. We are going to do it approximately. First we use the fact that n (h-1) is only going to be larger than n (h-2). Because as the height of the tree grows the number of nodes cannot reduce, it will only be more. So n (h-1) is at least as large as n (h-2). Then this implies what we had written earlier that is n (h) = n (h-1) + n (h-2) +1. This quantity is at least as large as 2n (h-2). Strictly larger because I also dropped the one. I have replaced this n (h-1) by n (h-2) and this (2n (h-2)) is what I get.

n (h) = n (h-1) + n (h-2) +1 > 2n (h-2)

This becomes the simple thing to solve, n (h) is more than 2n (h-2). This is what I will solve. So n (h) is more than 2n (h-2) and now n (h-2) is more than two times 2n (h-4). This implies the entire thing n (h) is more than 4n (h-4). Which implies that the entire thing is more than 8n (h-6). You understand how this comes n (h-4) is more than 2n (h-6) and so on, which will eventually take us to something like 2^i after i steps n (h-2i). Suppose I pick $i = \frac{h}{2-1}$, I am going to assume that this quantity is an integer. Let us assume that h was even to begin with, so this is an integer and for this value I will get n (h) $> 2^{\frac{h}{2-1}} n(2) = 2^{\frac{h}{2}}$ in which I replaced $i = \frac{h}{2-1}$. Recall n (2) was 2, so it becomes $2^{\frac{h}{2}}$. What does this say? We just argued that if your AVL tree has height h then it has at least $2^{\frac{h}{2}}$ nodes. That is at least so many nodes. What is the maximum number of nodes it can have? Something like 2^h *or* 2^{h+1}, one of those because it can be a complete binary tree.

(Refer Slide Time: 16:34)



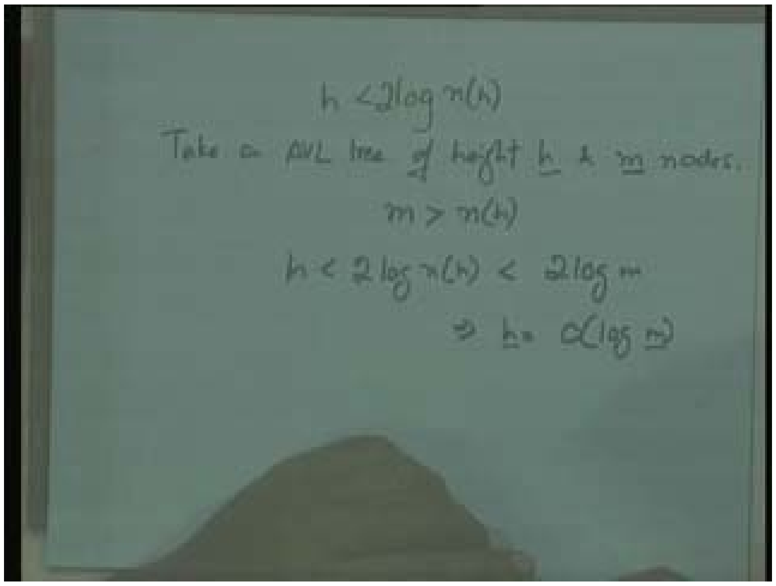
Height of an AVL Tree (2)

- Knowing n(h-1) >= n(h-2), we get
  n(h) = n(h-1) + n(h-2) + 1 > 2n(h-2)
  n(h) > 2n(h-2)
  > 4n(h-4)
  > 8n(h-6)
  ...
  > 2^in(h-2i)
- When i = h/2-1 we get: n(h) > $2^{h/2-1}$n(2) = $2^{h/2}$
- Taking logarithms: h < 2log n(h)
- Thus the height of an AVL tree is O(log n)

Suppose I were to take logarithms, what would I get? I would get h < 2log n. So n (h) is actually less than n because I have an AVL tree whose height is h and it has n nodes. Suppose I had an AVL tree of height h and n nodes then it will also satisfy this relation (h < 2log n (h)). It will satisfy the relation because n is only going to be larger than n (h). What was n (h)? n (h) was the minimum possible number of nodes. Any AVL tree on n nodes has height at most 2log n from this argument.

The h < 2log n (h) is what we argued after taking algorithms. Let me take a tree of height h and n nodes. So n is going to be larger than n (h) because n (h) is the minimum number of nodes that are possible in a tree of height h, n (h) is that quantity. This n is just a function, do not confuse this n with the number of nodes. You can replace this n with something else. n (h) is the minimum number of nodes in a tree of height h. What we argued was that h < 2log n (h). Take an AVL tree of height h and m nodes. Its height is h and it has m nodes in it. What does this implies? The m > n (h), this follows from our definition of n (h). We know that h < 2log n (h) which is then < 2log m. This implies h = O (log m).

(Refer Slide Time: 18:47)



$h < 2\log n(h)$

Take an AVL tree of height $\underline{h}$ & $\underline{m}$ nodes.

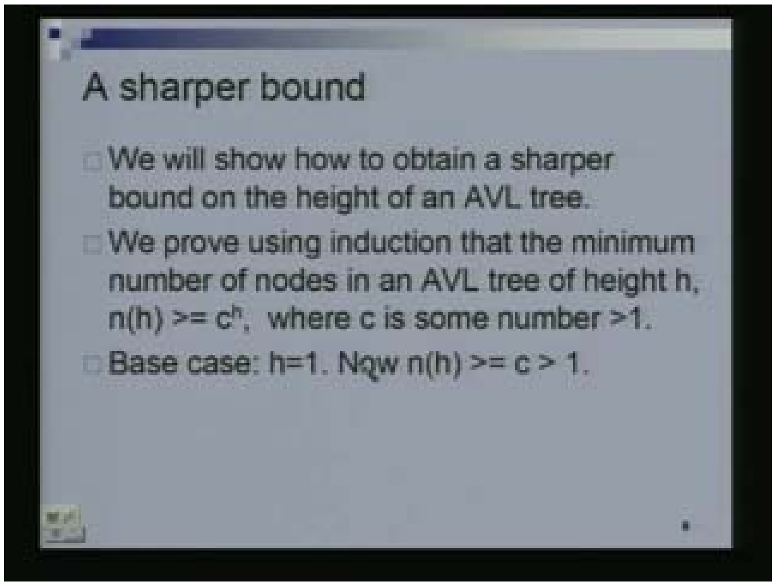
$m > n(h)$

$h < 2\log n(h) < 2\log m$

$\Rightarrow \underline{h} = O(\log \underline{m})$

The h is the height and m is the number of number of nodes. The height of an AVL tree on m nodes is less than two times log of the number of nodes. That is what being said here. We have shown that such a tree will have height no more than 2log n. The best possible tree could have height only log n if it were like a complete binary tree, very dense and every thing. But this has more height but not too much, just a factor of two more. Much better than having a height of an n. Let us try and solve this recurrence slightly better. This is more of an exercise also to show you how recurrences are solved. We did fairly crude analysis, we replaced n (h-1) with n (h-2) and then we did the steps and got the result. Let us try and get something better. It is just an exercise.

(Refer Slide Time: 21:19)



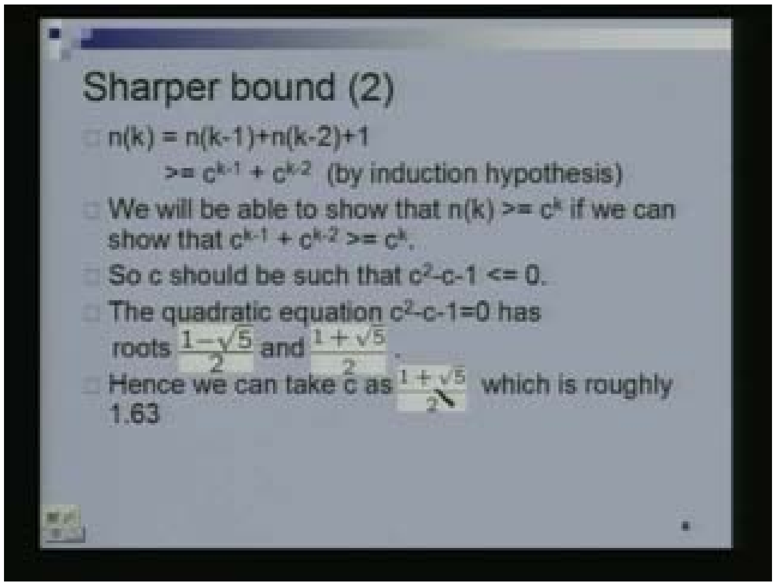
A sharper bound

- We will show how to obtain a sharper bound on the height of an AVL tree.
- We prove using induction that the minimum number of nodes in an AVL tree of height h, n(h) >= c^h, where c is some number >1.
- Base case: h=1. Now n(h) >= c > 1.

We will show how to get a sharper bound on the height of an AVL tree. The bound we obtained is 2log n. Let see if we can get something better than that. We are going to use induction and we are going to do a tighter analysis of the same thing. We are going to show that the minimum number of nodes in an AVL tree of height h which was n (h) is at least c times h that is c^h where c will be some constant more than one. What did we show in the previous slide? The n (h) was at least $2^{\frac{h}{2}}$. What was the c? It was $2^{\frac{h}{2}}$, there we showed a c of $\sqrt{2}$. Let see if I can get a higher c that is a larger c more than $\sqrt{2}$. What would be the way of doing such a thing? We will assume that n (h) is at least as large as c^h.

We are going to prove this by induction. We will figure out what c is later. We are proving a certain statement without actually knowing exactly what the statement is because I am not telling what c is. But you will see what the c has to be for the statement to be true. What is the base case? h=1, say n (h) is 1. This statement n (h) >= c^h is true at h=1. We have said the number of nodes is going to be at least as large as. I assume that I made a mistake, let us come back this base case again. We will have to perhaps redefine the height of a tree. I think we should have c^{h-1} or some thing. Suppose the claim is true for all h < k and lets try and prove it for h =k. We have to prove that n (k) >= c^k. We will come back to this base case in a minute. So recall this n (k) = n (k-1) + n (k-2) + 1 was our recurrence relation. Our induction hypothesis says that n (k-1) is at least c^{k-1}, n (k-2) is at least c^{k-2} and I have ignored this plus one. Actually I can say that this is strictly larger.

(Refer Slide Time: 25:16)



Sharper bound (2)

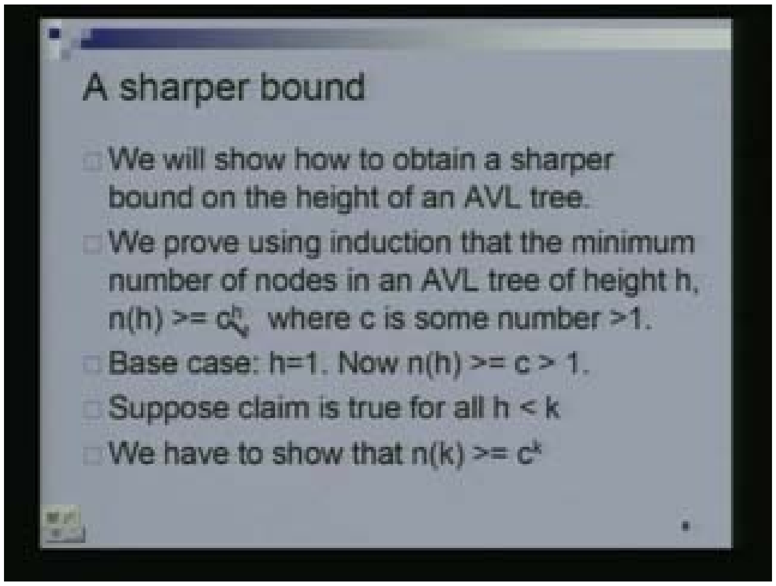
- n(k) = n(k-1)+n(k-2)+1
  $>= c^{k-1} + c^{k-2}$ (by induction hypothesis)
- We will be able to show that $n(k) >= c^k$ if we can show that $c^{k-1} + c^{k-2} >= c^k$.
- So c should be such that $c^2-c-1 <= 0$.
- The quadratic equation $c^2-c-1=0$ has roots $\frac{1-\sqrt{5}}{2}$ and $\frac{1+\sqrt{5}}{2}$.
- Hence we can take c as $\frac{1+\sqrt{5}}{2}$ which is roughly 1.63

I can show that n (k) is larger than c^k if I can show this quantity ($c^{k-1}+c^{k-2}$) is larger than c^k. This is what I have to show $c^{k-1}+c^{k-2}$ is lager than c^k. What should be the value of c so that this ($c^{k-1}+c^{k-2}>=c^k$) is true. I just cancel out the terms appropriately and I get $c^2-c-1<=0$. If c satisfy this ($c^2-c-1<=0$) then this ($c^{k-1}+c^{k-2}>=c^k$) will also be true. Why because I just multiply both the side by c^{k-2} and I would get exactly that. If this ($c^{k-1}+c^{k-2}>=c^k$) is true then n (k) which is larger than this ($c^{k-1}+c^{k-2}$) would also be larger than c^k. I just have to pick c which will satisfy this ($c^2-c-1<=0$). You all know how to figure out c which will satisfy this.

We will just solve this quadratic equation $c^2-c-1<=0$ and this has roots $\frac{1-\sqrt{5}}{2}$ *and* $\frac{1+\sqrt{5}}{2}$. This $\frac{1-\sqrt{5}}{2}$ is negative, so anything in between $\frac{1-\sqrt{5}}{2}$ *and* $\frac{1+\sqrt{5}}{2}$ would keep this c^2-c-1 less than zero. But I want as large as c as possible, so I will take $\frac{1+\sqrt{5}}{2}$ which is roughly 1.63. This quantity is also known as the golden ratio. Perhaps we will see this more often. This n (k) = n (k-1) + n (k-2) + 1 is not a fibonacci relation. If you add one to both sides, so n (k) would be with the fibonacci number minus one. You can also do that. We get a bound of roughly 1.63 that is c as 1.63. What is the mistake we have made? One thing is base case have not worked out. I guess this c^h was the wrong thing to pick. (Refer Slide Time: 26:28) It should not be c^h but may be c^{h-1}. So induction hypothesis should be c^{h-1}.

Let us take c^{h-1}, it will not make a difference. We take the c^{h-1} so precisely I am dividing out by c then the base would have also be satisfied. If h=1, you would have more than one which is the case. And sorry about the base case, for the other two also it will be okay. Because for h=2, n (2) is 2 and this would become c^{2-1} which is c. The c is less than 2 because we just argued it is 1.63. So please make that correction, we really require that the induction hypothesis is h-1. It will not make any difference on this (n (k) >= c^k) how ever. If this (n (h) >= c^h) become h-1 then this (n (k) >= c^k) will become k-1.

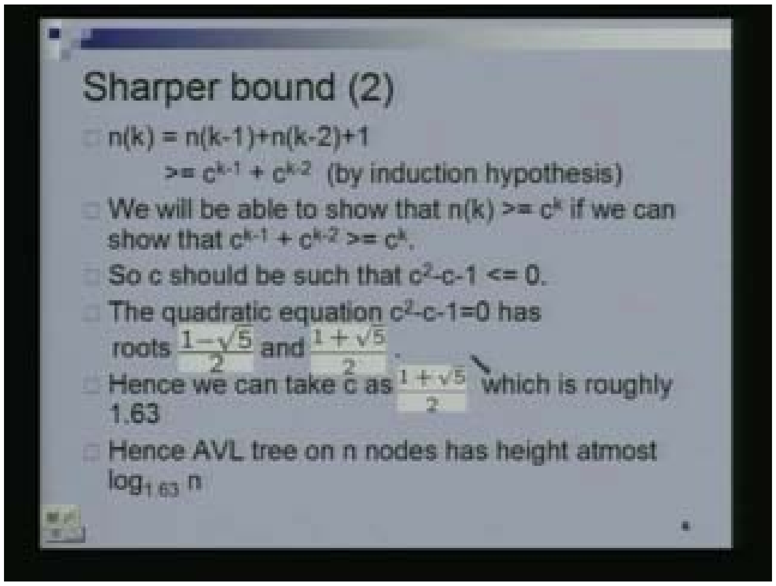
(Refer Slide Time: 27:05)



A sharper bound

- We will show how to obtain a sharper bound on the height of an AVL tree.
- We prove using induction that the minimum number of nodes in an AVL tree of height h, n(h) >= c^h where c is some number >1.
- Base case: h=1. Now n(h) >= c > 1.
- Suppose claim is true for all h < k
- We have to show that n(k) >= c^k

(Refer Slide Time: 28:05)

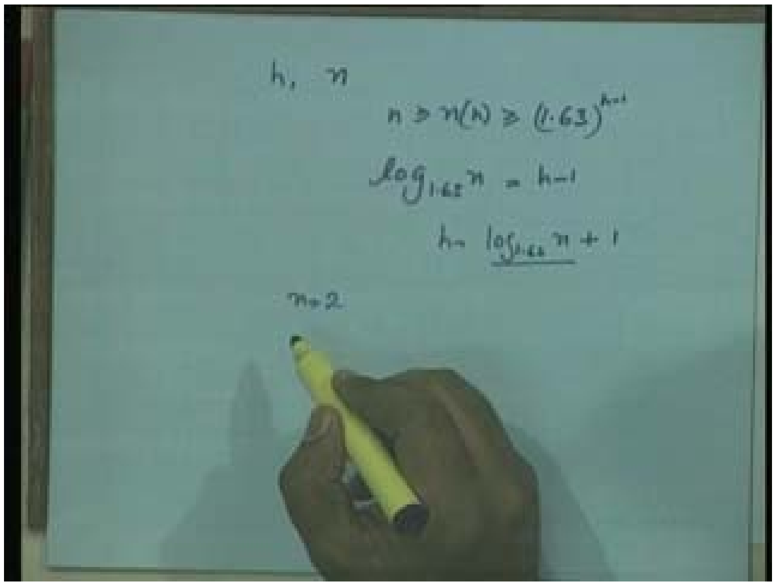


Sharper bound (2)

- n(k) = n(k-1)+n(k-2)+1
  >= $c^{k-1} + c^{k-2}$ (by induction hypothesis)
- We will be able to show that n(k) >= c^k if we can show that $c^{k-1} + c^{k-2}$ >= c^k.
- So c should be such that c^2-c-1 <= 0.
- The quadratic equation c^2-c-1=0 has roots $\frac{1-\sqrt{5}}{2}$ and $\frac{1+\sqrt{5}}{2}$.
- Hence we can take c as $\frac{1+\sqrt{5}}{2}$ which is roughly 1.63
- Hence AVL tree on n nodes has height atmost $\log_{1.63} n$

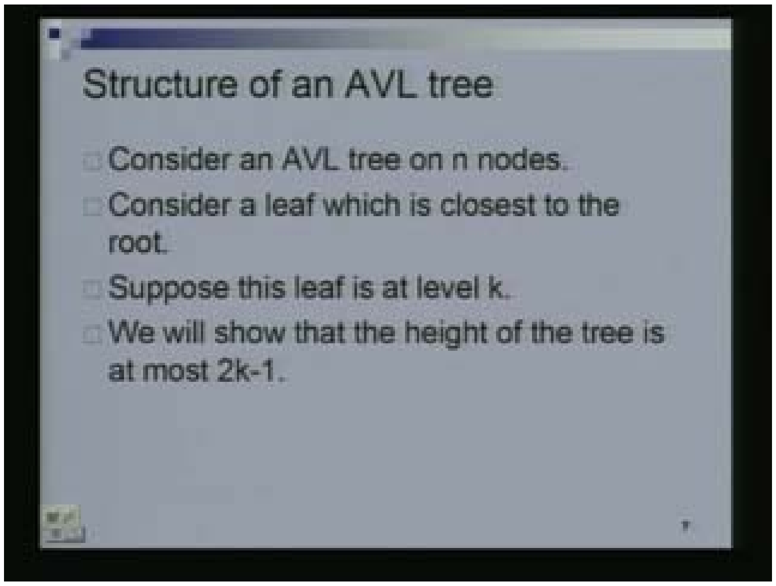
This ($>= c^{k-1} + c^{k-2}$) would continue as it is. This will become $c^{k-2} + c^{k-3}$. We have to prove this ($c^{k-1} + c^{k-2}$) is greater than or equal to c^{k-1}, every where there will be a minus one. So that you will still get the same ($c^2 - c - 1 <= 0$) quadratic in equality. The value of the c would still turn out to be the same. That is 1.63^{h-1} for n (h) please make that small correction. Thus the AVL tree on n nodes has height atmost $\log_{1.63} n$. We just do the same argument as before. I take a tree of height h and n nodes. We have just seen that $n(h) \geq 1.63^{h-1}$, this is the tree with smallest possible number of nodes. So n is going to be only larger than this that is $n > n(h) \geq 1.63^{h-1}$.

(Refer Slide Time: 29:44)


h, n
$n \geq n(h) \geq (1.63)^{h-1}$
$\log_{1.63} n = h-1$
$h = \underline{\log_{1.63} n} + 1$
n=2

Let us take log on both sides, we get $\log_{1.63} n$ =h-1. I am just using the definition of log. I am taking $\log_{1.63} n$, so I will get h-1. This implies h= $\log_{1.63} n$ +1. We are able to prove this kind of a sharper bound. This equation also works for n=2, so that was our base case. Let us continue. I have shown you the 2 ways of solving this same recurrence. One was the much simpler way, actually both are very simple. The 2nd technique is also used quite often. You make a guess on what you think the right value should be. Then essentially you verify that. We said that suppose the right value is some c^h and then you figure out what your c should be. You can get something better, earlier we had $\sqrt{2}$ that is 1.414 and we could update to 1.63 by using this kind of a technique.

(Refer Slide Time: 31:20)



Structure of an AVL tree

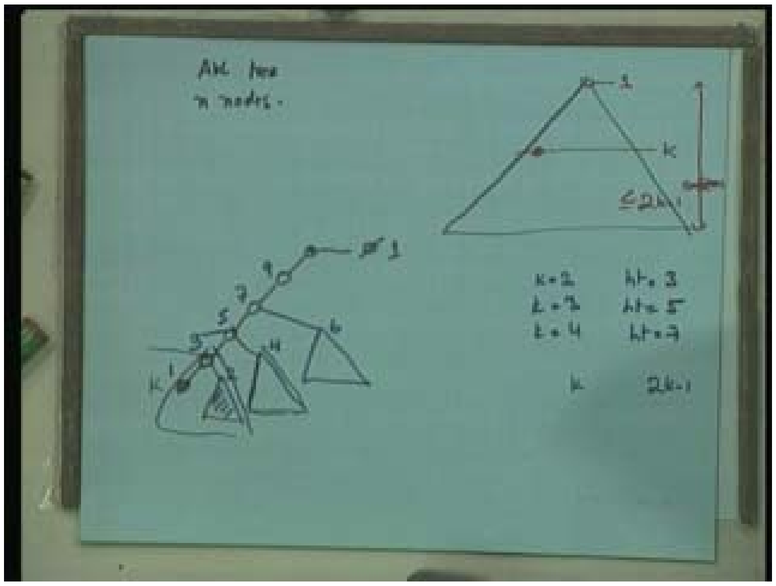
- Consider an AVL tree on n nodes.
- Consider a leaf which is closest to the root.
- Suppose this leaf is at level k.
- We will show that the height of the tree is at most 2k-1.

Let us look at the structure of an AVL tree in detail. Once again I have an AVL tree on n nodes. Let me take the leaf of this tree which is closest to the root, which means whose level number is the smallest among all the leaves. Suppose this leaf is at level k. We can show that the height of the tree is at most 2k-1. This requires the proof and let us do that. I have an AVL tree which has n nodes in it, although the number of nodes in the tree is not going to be particularly important. This is some tree, I took that leaf of the tree which is closest to the root. Suppose the red dot is the leaf which is closest to the root. We said that it is at level k. So the other leaves could be at this level or could be below.

In this class for AVL tree we work with level starting with one. It does not make a big difference, let us say we start with level one. We are going to prove that the height of this tree is at most 2k-1. So the height of this tree is $\leq 2k-1$ that is what we will prove. Let see why. I will draw this picture again. This is the leaf which I have colored red is at level k and it is the one which is the closest to the root. From the node which is next to k, there will be some sub tree hanging out. From the next node also there will be some sub tree hanging out and so on.

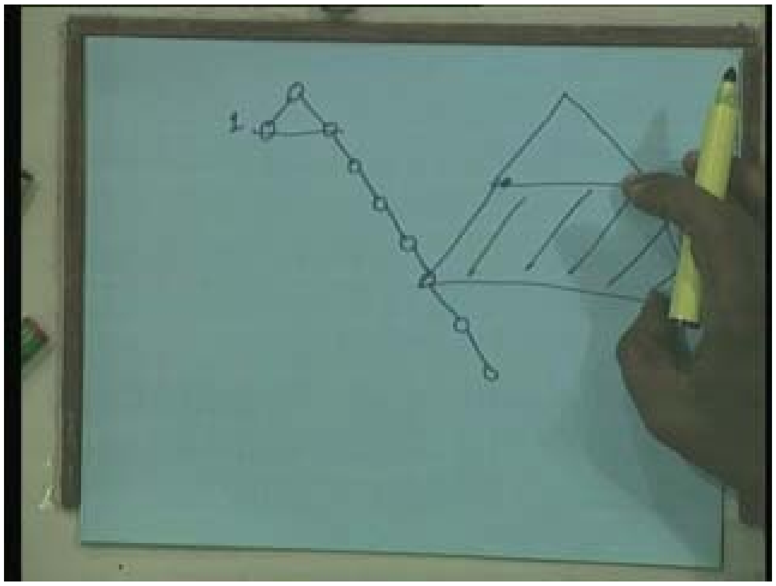
The first node is my root at level 1. Let us look at this node which is at level k-1. What is the height of this node at level k-1? It has one child and this child has height one. The heights are in blue. This means this sub tree at level k-1 can have height at most 2. We want to get as larger height as possible for this tree. Whenever we say at most 2 will just take the largest value. This can have height 2, if this has height 2 then what is the height of this node? This sub tree will have height 3. If this sub tree has height 3, what is the largest height that the next sub tree can have? It can have 4. What is the height of this node? It is 5.

(Refer Slide Time: 35:45)



What is the maximum height this sub tree can have? It is 6 and this node would be 7, 9 and so on. What will be the height of the root? In general given that this was k, just figure it out, it should be 2k-1. If it was just till the node 3 then it is basically k=2 height=3. If k=2 then the height =3, if k=3 then the height was 5. If k was 4 then height was 7 and so on. For arbitrary k this is 2k-1. It is a very simple argument which means that this entire tree can be no taller than 2k-1, if the closest leaf was at level k. This is the property of AVL tree and not a property of any arbitrary binary tree.

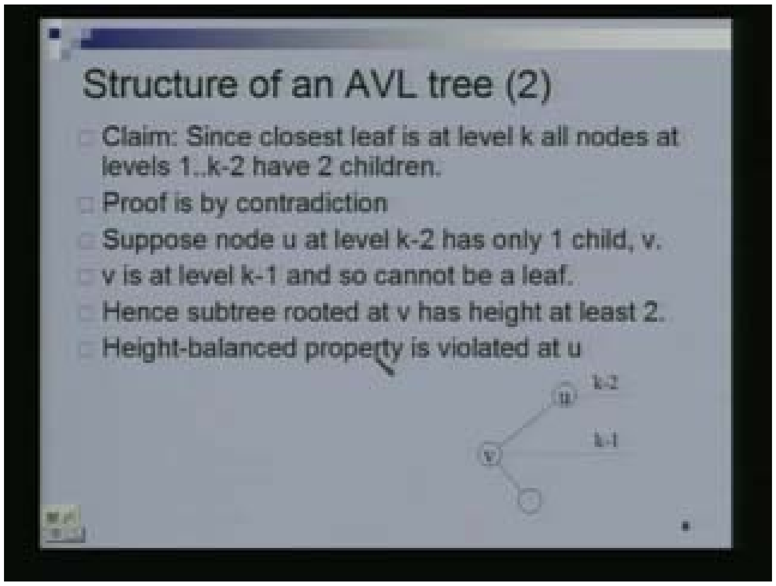
(Refer Slide Time: 37:03)



In an arbitrary binary tree you might have leaves at any level. But the height of the tree could be as bad as you wanted. Here is a leaf at level 1, (hindi). (Refer Slide Time: 36:30) But for an AVL tree if there is a leaf at level k then the height of the tree can not be more than 2k. So in any AVL tree basically all our leaves will be in the shaded part of the above slide. This band whose width is as large as this roughly and both of them was k so I am ignoring that. I will just come back to this in a minute. We just argued that if the closest leaf is at the level k then the height of the tree is no more than 2k-1. (Refer Slide Time: 37:35)That is the largest possible height the tree can have.

Let us make another claim. If the closest leaf is at level k then all nodes at level 1 through k-2 have 2 children. Every node on these 1st k-2 levels should have 2 children. Why have I said k-2 and not k-1? Let us prove this by contradiction. What do we want to do contradict? Let us take some node at level k-2 which has only 1 child. The picture is given in the below slide. I have a node u at level k-2, it has only 1 child which is at level k-1. I have shown a node at level k-2 but the same argument would apply to any node at 1 through k-2. So v is at level k-1, it cannot be a leaf because our closest leaf was at level k.

(Refer Slide Time: 39:13)



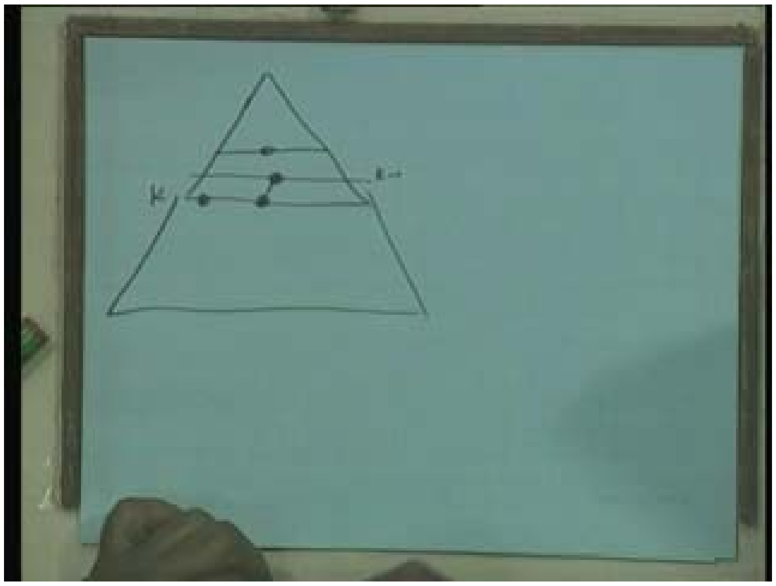
Structure of an AVL tree (2)

- Claim: Since closest leaf is at level k all nodes at levels 1..k-2 have 2 children.
- Proof is by contradiction
- Suppose node u at level k-2 has only 1 child, v.
- v is at level k-1 and so cannot be a leaf.
- Hence subtree rooted at v has height at least 2.
- Height-balanced property is violated at u

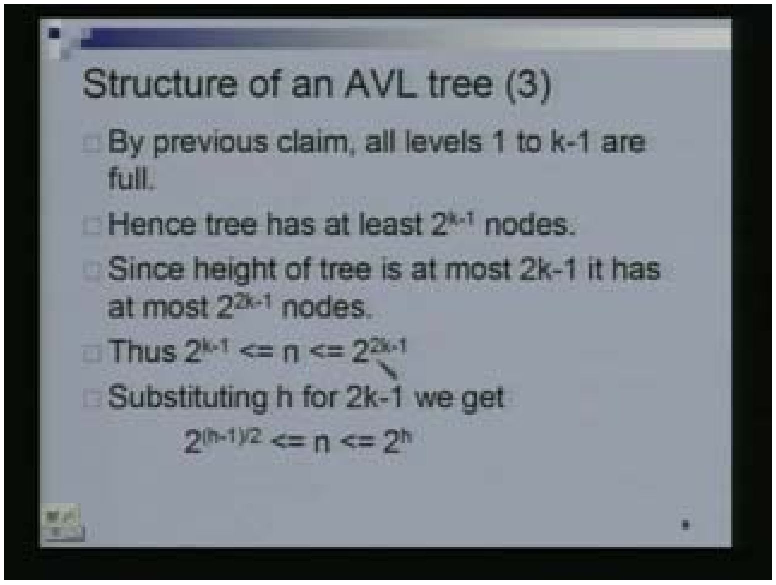
So it has to have another child. I have shown only 1 but it can also have 2 children. But this u has only 1 child. So sub tree rooted at v has height at least 2 because this should have 1 child, it cannot be a leaf. It has height at least 2 while the right sub tree here has height zero because there is nothing there. So we have a height imbalance at this node u. The height balance property is violated at u. Every node on these levels 1 through k-2 should have 2 children. At level k-1 how ever there can be nodes with only 1 child.

This is level k, of course the tree extends. The dot on the left side is the level at which the closest leaf is situated. At level k-1, I can have a node with only 1 child and that child is the one which is in the middle. And provided it would not have any more descendants. It need not have descendants because it can be a leaf. This is completely okay but if it had more descendants then we would again have a problem in height balance property. This is okay which means that the node in the level in k-1 can have only 1 child. But everything which is in the 1st place should have at least 2 children. We said every node at level k-2 should have 2 children which means levels 1 through k-1 are full. It means they have as many nodes as possible on that level in a binary tree. This is after all a binary tree. So they are full.

(Refer Slide Time: 40:58)



(Refer Slide Time: 42:24)



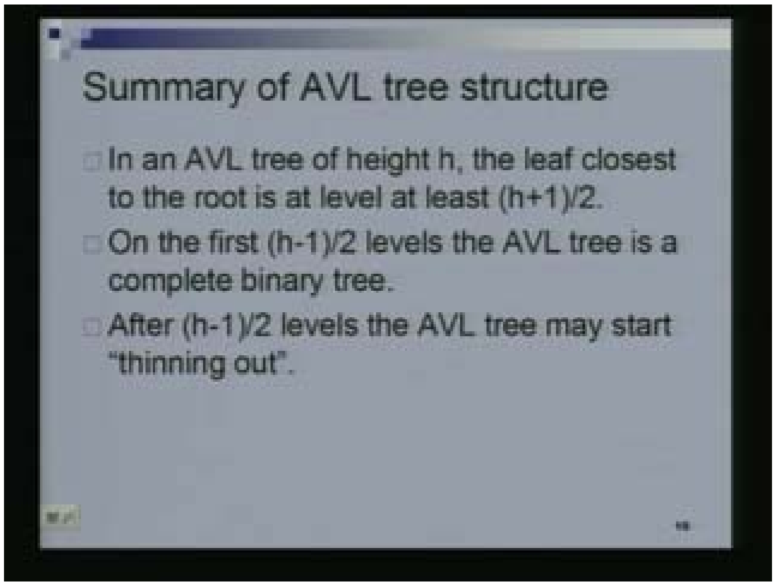
Structure of an AVL tree (3)

- By previous claim, all levels 1 to k-1 are full.
- Hence tree has at least 2^{k-1} nodes.
- Since height of tree is at most 2k-1 it has at most 2^{2k-1} nodes.
- Thus $2^{k-1} <= n <= 2^{2k-1}$
- Substituting h for 2k-1 we get

  $2^{(h-1)/2} <= n <= 2^h$

What does that mean? That means the tree has at least 2^{k-1} nodes. We also argued recall that the height of the tree is at most 2k-1. If the height of the tree is 2k-1 then it has at most 2^{2k-1} nodes. This implies the number of nodes in the tree which was n is between 2^{k-1} and 2^{2k-1}. Since we have been using h for the height, let us substitute h for 2k-1. Let us see how this equation would look like. This (2^{2k-1}) becomes 2^h and this (2^{k-1}) becomes $2^{\frac{(h-1)}{2}}$. This is the same thing I am showing you again. What is this saying? If you have an AVL tree of height h then it has at least $2^{\frac{h}{2}}$ which we had shown earlier. Now we are showing $2^{\frac{(h-1)}{2}}$ just roughly the same thing $\sqrt{2^h}$ nodes, all though we have proved the sharper bound. I am coming back to the older bound. The point is it has an exponential number of nodes, it has number of nodes which is some constant c^h an exponential. Because that gives the logarithmic height property. This is actually a third way of proving that the height of the tree is only log n. You can also use this as a proof. This did not require solving a recurrence relation. The other 2 methods we saw while solving the recurrence relation. But the sharpest bound we have seen so far is c^h that is 1.63^h.

(Refer Slide Time: 45:31)



Summary of AVL tree structure

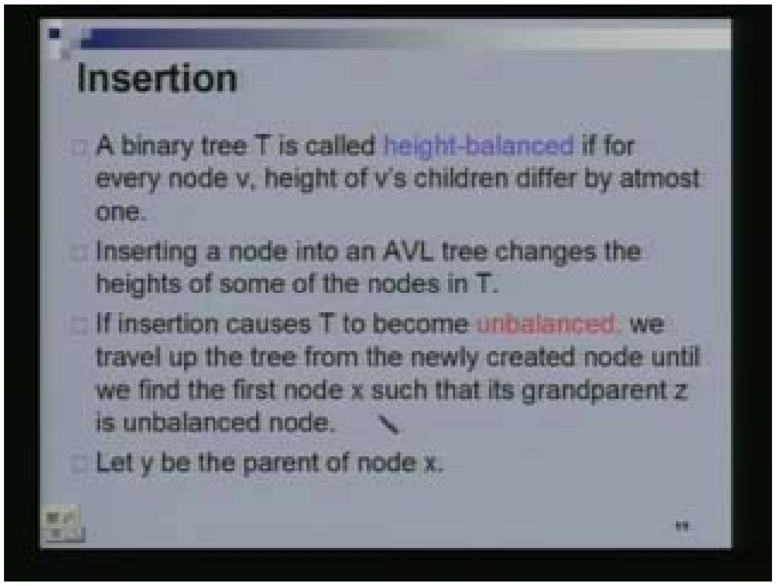
- In an AVL tree of height h, the leaf closest to the root is at level at least (h+1)/2.
- On the first (h-1)/2 levels the AVL tree is a complete binary tree.
- After (h-1)/2 levels the AVL tree may start "thinning out".

Let us summarize what we have seen as the structure of an AVL tree is concerned. If the height of an AVL tree is h then the closest leaf can be at level $\frac{h+1}{2}$. I have just changed things around, when I said when the closest was at k then the height was 2k-1. If the height is h, suppose I give you an AVL tree on n nodes of height h then the leaf which is closest to the root is actually pretty far from the root. It is atleast half the height away, it is at least $\frac{h+1}{2}$ away. It does not require a proof, I am just rewording what I have said earlier. We also saw that on the first $\frac{h+1}{2}$ levels the AVL tree is a complete binary tree.

This is what an AVL tree looks like essentially. For the first half levels it is complete, very dense and then it starts thinning out. So it turn the tree around with the root at the bottom so initially it is dense and then it thins to the full height. But the fact that it is very dense for the first edge by 2 levels means it has a lot of nodes. It is a complete binary tree so it has $2^{\frac{edge}{2}}$ nodes straight away. That means that the height can not be too large, if I had n nodes the height can not be more than 2log n. Once again I have said that if number of nodes in the AVL tree is at least just this fact, since it is a complete binary tree on $\frac{h-1}{2}$ levels it has at least $2^{\frac{(h-1)}{2}}$ and at most 2^h nodes because that is the height of the tree.

This is the useful structural fact to keep in mind about AVL trees. Although we will not use it for any of our algorithms. But it just gives you some intuition of what the tree is and why is that this tree has only a logarithmic depth. We have looked at this height balance property, we said if this height balance property is there then it is nice the height of the tree is only algorithmic. We want to say that all our operations are only logarithmic because we still want to say that you can do a search, insert and delete in log n times. Search is easy there is no problem with search because after all it is a binary search tree. Forget the height balance property, it is just a binary search tree so you just do search as you do in a binary search tree. How much time will you take? Proportional to the height, order h. Height is log n so you will take only log n time. That is the best you can do in some sense.

(Refer Slide Time: 49:16)



Insertion

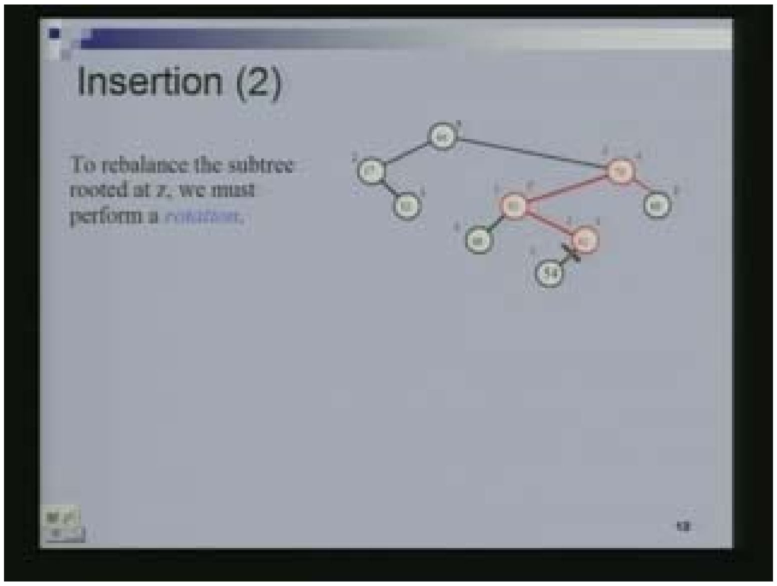
- A binary tree T is called height-balanced if for every node v, height of v's children differ by atmost one.
- Inserting a node into an AVL tree changes the heights of some of the nodes in T.
- If insertion causes T to become unbalanced, we travel up the tree from the newly created node until we find the first node x such that its grandparent z is unbalanced node.
- Let y be the parent of node x.

Suppose you were to try an insert. When you are going to do an insert what can go wrong. Recall for the tree to be height balanced, if the difference in the heights of its children is at most a one. When I insert a node it can change the height of some nodes and as a consequence the height balanced property might get violated. The first step of insertion would be the same as we did in the case of a binary search tree.

How did we insert in a binary search tree? First you find the position. How do you find the position? You will just search for that element that you are trying to insert, that will tell you where the position is and just put the node there. And then you start marching up back to the route by following the parent pointers. As you march up you keep updating the heights of the various nodes you encountered because these are the only nodes whose heights could have changed and no one else. We will look at this again in more detail perhaps in the next class. I am just giving you the flavor of what needs to be done when we are doing an insertion. These are the nodes whose heights are going to change.

So we are going to the first place where the height change appears, where the height imbalance happens. We are going to only start from the node where we inserted and move up the tree towards the root. Basically we keep going parent, parent till we hit the root. On this path that we follow, we find the first node which has the height imbalance property. Suppose that node is called z and its grandchild is called x. Let me skip this part and y is the node in the middle. So I think it is best if I show you the picture and that will give you an idea.

(Refer Slide Time: 51:13)



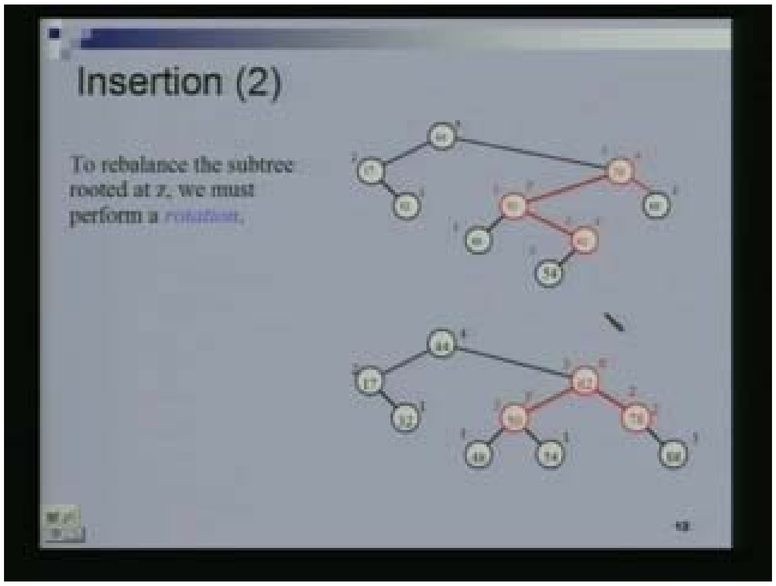
Insertion (2)

To rebalance the subtree rooted at z, we must perform a rotation.

Suppose the 1st one was my tree, forget this empty node which is the last node for now. This was my tree originally. If this was my original tree, then is that an AVL tree? Height balance is satisfied in the node 50 because 48 is one and this 62 is one. The last node is not there, forget this type of node. This 78 is also height balanced because this 50 is 2 and this 88 is 1. This is also height balanced because this is 1 and this is 0. (Refer Slide Time: 49:46) This is height balanced because this is 2 and this is 3 initially. (Refer Slide Time: 49:47- 49:52) But now suppose I went and inserted a node 54 which came in here. The 54 would come here, I go right, left and then right here and left here. (Refer Slide Time: 49:56- 50:03) Now the height balance property is violated. What I am going to do? I am just going up the tree towards the root. Is the height balance property violated here? (Refer Slide Time: 50:16) No it is not.

This is one this is zero. Is it violated here? (Refer Slide Time: 50:21) No, 1, 2. This is height 2, this is height 1. (Refer Slide Time: 50:27- 50:30) It is not violated here. Is it violated here? Yes because this is now 3 and this is 1, so these numbers are the new heights. (Refer Slide Time: 50:31- 50: 37) This is 3. (Refer Slide Time: 50:40) So this 78 is the first node at which the height balance property is violated. We call this node 78 as z, its child will be y and its grand child will be x. We wonder which child of this node will be y. The child on the path that we have taken.

(Refer Slide Time: 51:20)



Insertion (2)

To rebalance the subtree rooted at z, we must perform a rotation.

And now we need to do something to this tree to make it height balanced again. This is not height balanced tree. All the things we said about log n will go out of the window if you leave the tree like this. What are we going to do? We are going to do a kind of rotation operation and this 2nd picture in the above slide will become my new tree. In some sense what I have done is, I have moved 62 up and moved this 78 down and this 50 was here. It looks a bit mysterious. That is what we are going to do in the next class. Understand how this rotation operation is done. So as you can see now the height balance property is not violated at any node. It is not clearly in the node 50, its not here in 62 and also in 78. Both of them (50 and 78) have height 2. The 62 is at height 3 and this 17 is at height 2, so it is not violated. This is still a binary search tree with the same keys as before, we will not change the keys.

There are other ways also but you want an automated way of doing it, you do not have to draw the picture and then figure out what rotation have to be done. You will be able to do this program. This is what we are going to do in the next class. Look at insertion and look at how to do these rotations so that the height balance property is retained even after insertion. So we will look at both insertion and deletion in the next class. So in todays class we looked at AVL trees. We saw how AVL trees are defined and actually we proved a bound of $\log_{1.63} n$ as the height of an AVL tree. We spent a lot of time figuring out how to solve that recurrence relation. We saw 2 ways solving that recurrence relation. We also looked at some structural property of the tree which also proved a similar bound and the height of the tree. With that we will end today’s class.