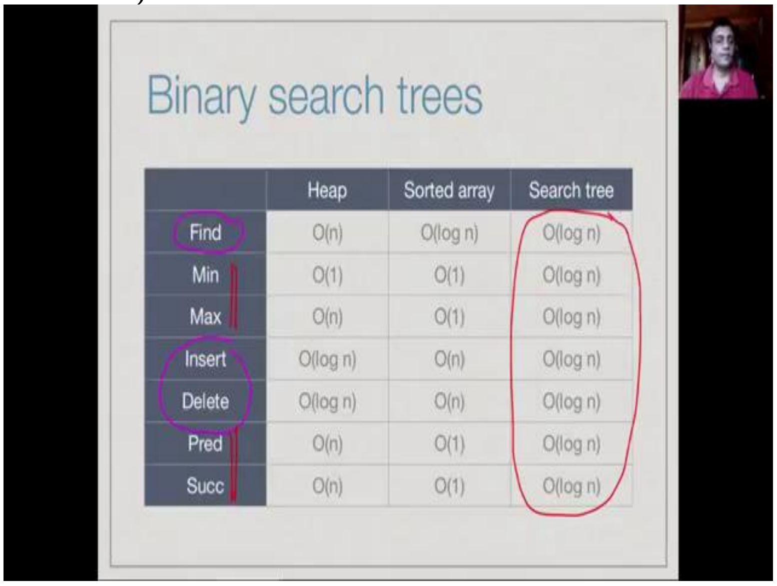
Design and Analysis of Algorithms, Chennai Mathematical Institute
Prof. Madhavan Mukund
Department of Computer Science and Engineering,

Module – 02
Lecture - 40
Balanced Search Trees

In the previous lecture, we looked at operations on search trees. We claim that these were efficient that we could maintain balance. So, let us see how we can keep search trees balanced.

(Refer Slide Time: 00:12)

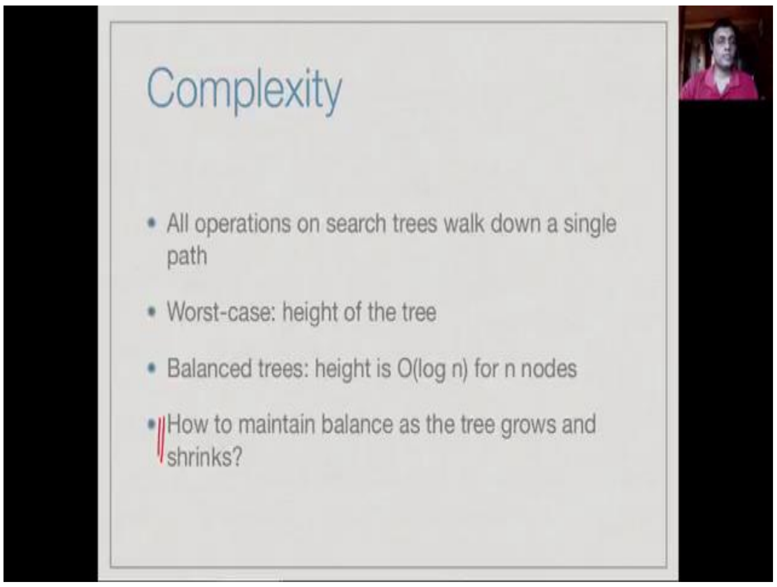


Binary search trees

	Heap	Sorted array	Search tree
Find	O(n)	O(log n)	O(log n)
Min	O(1)	O(1)	O(log n)
Max	O(n)	O(1)	O(log n)
Insert	O(log n)	O(n)	O(log n)
Delete	O(log n)	O(n)	O(log n)
Pred	O(n)	O(1)	O(log n)
Succ	O(n)	O(1)	O(log n)

So, recall that we are looking at these 7 operations, we want to able to search for a value, you want be able to insert and delete values, you also want to be able to compute the minimum and the maximum value in a tree and also find the predecessors and successor of the given value and all of these we claimed would be order log n provided the tress is balanced.

(Refer Slide Time: 00:34)

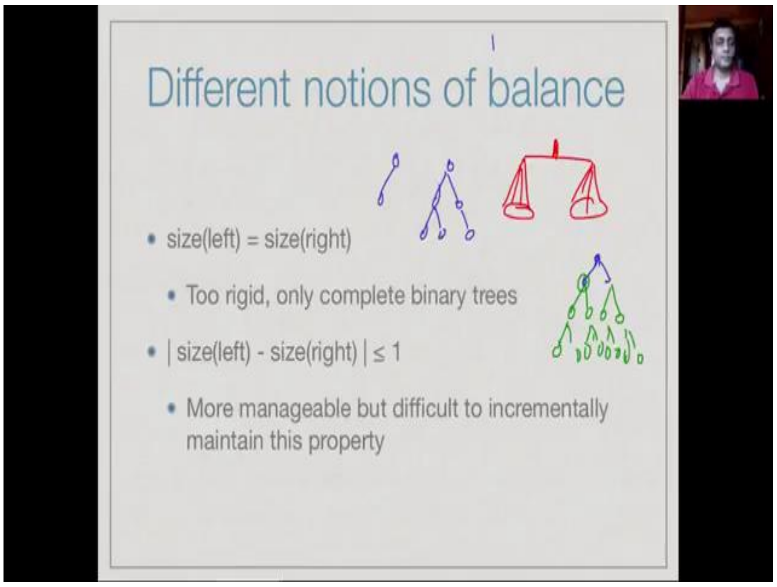


Complexity

- All operations on search trees walk down a single path
- Worst-case: height of the tree
- Balanced trees: height is O(log n) for n nodes
- How to maintain balance as the tree grows and shrinks?

So, thus because all of the operations as we implemented them, we will walk up and down a single path and so the worst case would be the height of the tree and in our balanced tree the height will always be logarithmic in the size of the tree. So, today's this lecture the goal is to explain, how to maintain the balance as the tree grows and shrinks.

(Refer Slide Time: 00:54)



Different notions of balance

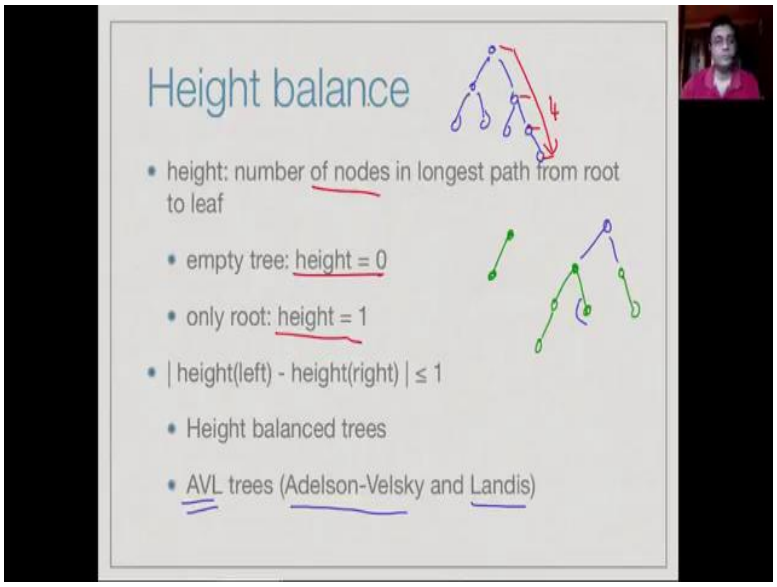
- size(left) = size(right)
  - Too rigid, only complete binary trees
- | size(left) - size(right) | ≤ 1
  - More manageable but difficult to incrementally maintain this property

So, trees there are many different notions of balance in a tree, so essentially a balance we should think of it is like a physical balance. So, if you go to an old style vegetable seller, when they will have this kind of a balance and then you want at any point if you hold up a tree by it is root, the two side should be balanced, they should be equal. So, the most direct notion of balance is that the two are exactly that is the number of nodes at the left is equal to the number of nodes in a right for every node.

Then, it is easy to see that you when you get a complete binary trees, so example you could have a tree which has this structure or if you extend it one more level, then for every node in the left you must extended on the right, but then these must also be balanced, so you must definitely complete this. So, you can have a tree up to 3 levels which is completely filled or up to 4 levels and so on. So, if you want this exact size balance, then it is very restricted.

So, you might have a little bit more flexibility, you might say there we do not want to be exactly equal, we may be wanted it to be at most one off, then you could have structures for instants like this, where you have only a left or you have at this point a left and right put you have only array. These standards structures, now which are not complete binary trees become balanced in this notion. So, this allows us more flexibility and we can get more trees this way, but it is difficult to maintain this property incrementally as we do inserts and deletes. So, we will go for different notion of balance.

(Refer Slide Time: 02:25)



Height balance

- height: number of nodes in longest path from root to leaf
  - empty tree: height = 0
  - only root: height = 1
- | height(left) - height(right) | ≤ 1
  - Height balanced trees
  - AVL trees (Adelson-Velsky and Landis)

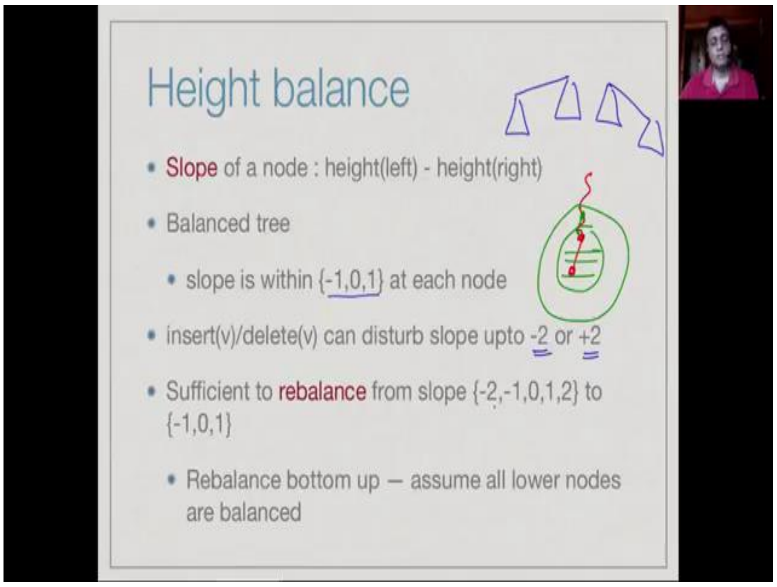
Notion of balance that we will use is not with respect to size, but with respect to height. So, we will say that the height is a number of nodes from the root to a leaf. For example, if I have a tree which looks like this, then here the height is 1, 2, 3, 4 because on this path we have 4 nodes. So, the heights become 4 it is a length of the longest path measured in terms of nodes, the reason we measured in terms of nodes is, then we can distinguish easily, the empty tree from the tree with only are root.

If you measured in terms of edges, the tree with only a root will have height 0, because there are no edges and so would be the empty tree. Whereas, if we measure it in terms of nodes, then the empty tree has height 0 and the tree with only the root has height 1. So, we can distinguish these two. And now in keeping with our earlier relaxation of the size condition, the height balance tree will be one where the height of the left and the height of the right differ the at most 1.

Now, this is more relaxed in the previous thing. For example, now of course, I could start with a height balance tree like this. And then, I could now connect this to form a height balance tree like this and now this which is height 3 tree I can connect with a height 2 tree and form a height balance tree which looks like this. So, the height of the left sub tree is 3, height of the right sub tree is 2 in this recursively the height of left sub tree is 2, the height of the right is sub tree is 1 and so on.

So, we could have things which look quit difference, so size here for instant size is 4 and size is 2. But, nevertheless you can kind of compute that the size even in this case will be exponential in the height or rather the height will be logarithmic in the size. So, these trees are called AVL trees the named after the two people who independently invented them one person called Adelson-Velsky and independently Landis. So, an AVL tree is a height balanced tree which says that at every node the height of the left and height of the right sub trees differ by at most 1.

(Refer Slide Time: 04:53)



Height balance

- Slope of a node : height(left) - height(right)
- Balanced tree
    - slope is within {-1,0,1} at each node
- insert(v)/delete(v) can disturb slope upto -2 or +2
- Sufficient to rebalance from slope {-2,-1,0,1,2} to {-1,0,1}
    - Rebalance bottom up — assume all lower nodes are balanced

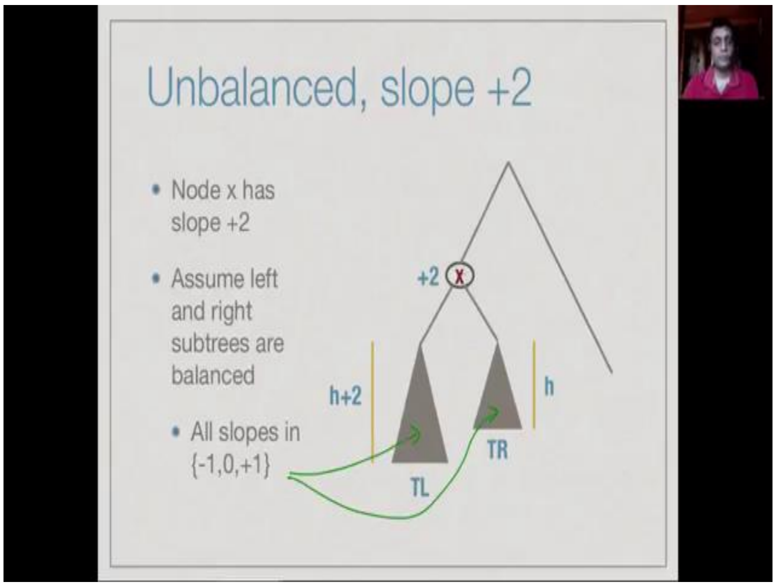
So, let us refer to the difference between the height as just the slope, so we have a intuitively in our pictures. So, if it is unbalance then thing is treated, so we could have till this way or till this way, so we call this the slope. So, we let us say this slope is height of the left minus height, so the height of the left is less than the height of the right, then you have a positive slope. If the height of the left is bigger than the height of the right, then you have right, left is smaller than right you have negative slope, left is bigger than right you are positive slope.

So, in a balanced tree since the height difference absolute value must be 1, you can only have three possible slopes throughout the tree, either there is no slope they are exactly the same or it is minus 1 or plus 1. Now, if you can argue very easily that if the current value is of the slope some minus 1 plus 1, when you delete a node, you can reduce one of the heights by 1. So, the height difference can go from 1 to 2 or when you increase you can make the height difference, again go from 1 to 2.

So, the new slope after a single insert or a single delete can be at most minus 1 or plus 2, minus 2 or plus 2. So, what we will end up to do what we will try to do is that whenever we do an insert or a delete we will immediately try to rebalance the tree. So, we would have a single disturbance from minus 2 or plus 2 it will never become very badly unbalance and we will immediately restore the balance to within minus 1 to plus 1.

So, you will do this rebalancing we will also do this rebalancing bottom up, so what happens we will be do an insert, if you remember is that we go down and we find a place to insert. So, this point we add a new node, so therefore now at this point that could be some imbalance, so we fix it, then we will go back to the up this path and we will go there and we will fix the path here, but at this point you will assume that the tree below has been balanced. So, whenever we rebalance the slope which is outside the range, you will assume that the sub trees below that are already balanced, because this balancing as we will see is going to be done bottom up.

(Refer Slide Time: 06:59)



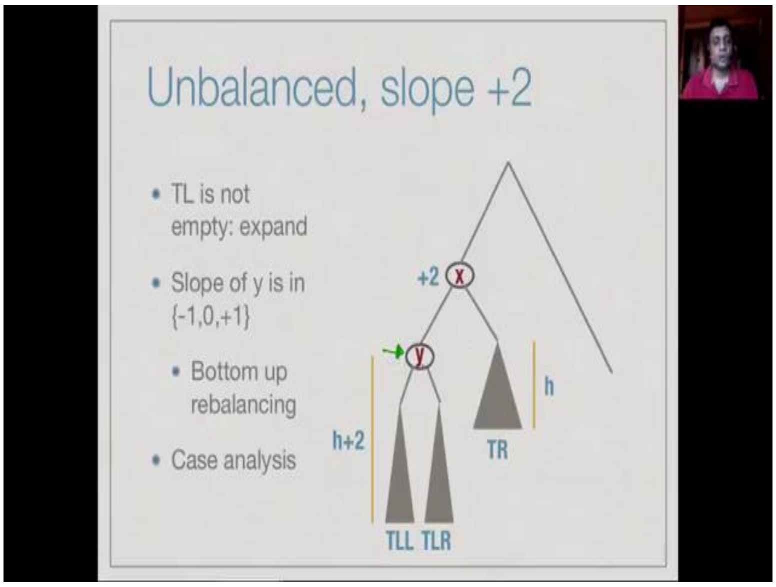
Unbalanced, slope +2

- Node x has slope +2
- Assume left and right subtrees are balanced
  - All slopes in {-1,0,+1}

So, here is a typical situation that we would reach after a single operation which removes the balance. So, we might have a node which has slope plus 2 or minus 2, so let us look at plus 2 minus 2 turn out be symmetric. So, we have a node which we call x which has slope plus 2 and what it means is, it has a left tree and right tree. Such that, the height of the left tree is 2 more than the height of the right tree, remember this slope is right or left minus right or left, so h plus 2 minus h will be 2.

Now, recursively we are going to assume that all the slopes here and here are at most plus 1 or minus 1. So, we are assuming that everything below this has been fixed and the only in balance in this sub tree at x is x itself.

(Refer Slide Time: 07:50)



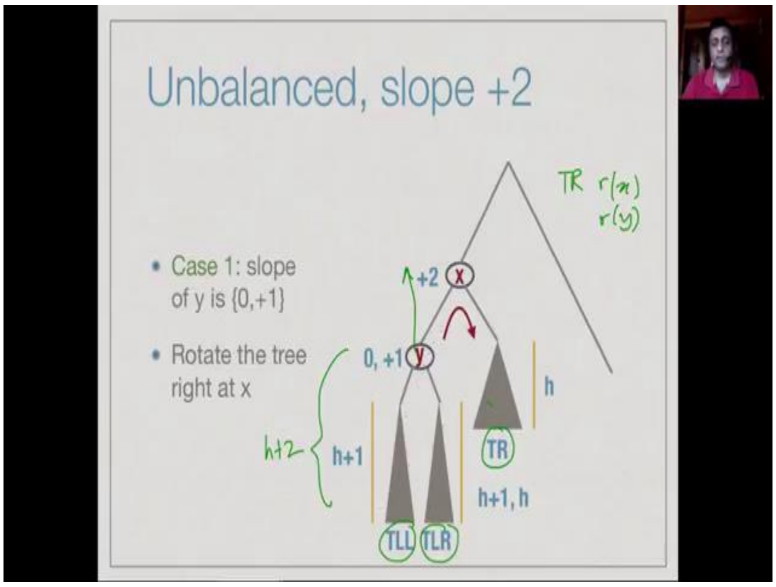
Unbalanced, slope +2

- TL is not empty: expand
- Slope of y is in {-1,0,+1}
  - Bottom up rebalancing
- Case analysis

So, now since the left has height h plus 2, it has height at least 2, h can be at most at least smallest h can be 0, so it has height at least 2, so there at least 1 node here. I mean 2 means that there are at least 2 nodes here, so we have at least 1 node in particular. So, we will expand this by exposing it is root and the root will have in general 2 sub trees, so now this whole thing as height h plus 2.

So, we will now look at this new node that we have expected. So, this slope is minus 1 0 or plus 1 and we are going to do some bottom up rebalancing, we are assuming everything below it is case. So, I have going to do some case analysis based on what is the slope of y.

(Refer Slide Time: 08:41)



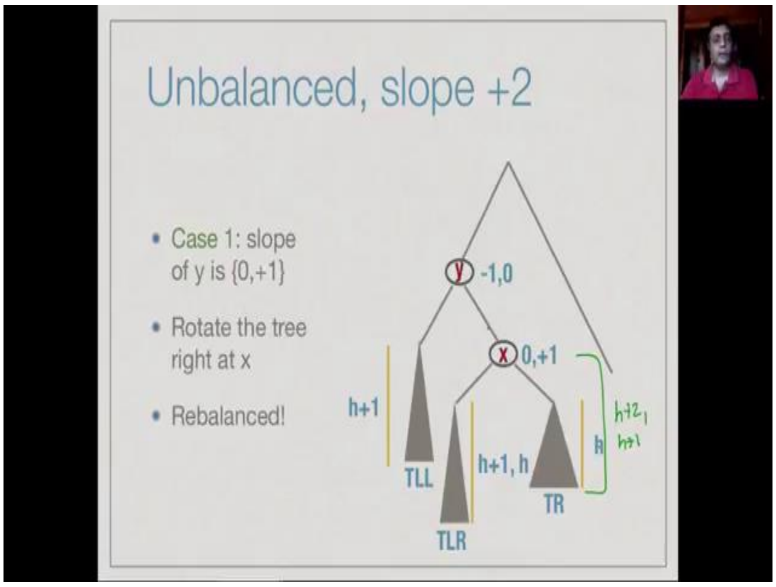
Unbalanced, slope +2

- Case 1: slope of y is {0,+1}
- Rotate the tree right at x

So, let us first look at the situation where the slope of y is either 0 or plus 1, so if it is 0 or plus 1 it means that so remember this whole thing was h plus 2 of which 1 node is here, so it is left child must be at least h plus 1 and because it is slope is 0 or plus 1, the right child is either h plus 1 in case slope is 0 or it is h incase the slope is plus 1. So, now this is the current situation as we have it with an unbalanced node x everything below is balanced. But we have just come to a situation where we try to analyze what is the situation behind.

So, x is got a balanced unbalance of plus 2 and below it we have why which whereas assuming is either 0 or plus 1. So, now we do this rotation, so we take this tree and we kind of hang it out by y and we reattach things. So, in this rotation when you hang it out by y, x comes down and now we look at this sub trees, so we have the 3 sub trees, we have TLL, TLR and TR. So, TR is to the right of x and it is also to the right of y, so it is the right of both, TLR is to the left of x to the right of y, TLL is to left of y, left of x.

(Refer Slide Time: 09:57)



Unbalanced, slope +2

- Case 1: slope of y is {0,+1}
- Rotate the tree right at x
- Rebalanced!

So, if you go there we find that TR is to the right of both, TLR is to the left of x right of y and TLL is to the left of both. So, we hang up these trees, so now all the values we can verify will be currently organized as a search tree issue. But, now if you look at the slopes, we have just inherited this slope some what we knew that the slope of TLL of h plus 1, TLR is h plus 1 or h and TR is h.

So, this means that if I look at this over all height at this point, it is either h plus 1 or at most h plus 2, so this is h plus 2 or h plus 1. If it is h plus 2, then the height slope at y is minus 1, if it h plus 1 then both sides at h plus 1 slope at y is 0 and if you look at x, here we have h plus 1 and here we have h, so the difference is either 0 or plus 1. So, x is now balanced, y is balanced and by assumption inductively all the grey sub trees are balanced. So, by one right rotation, we have rebalanced the tree.