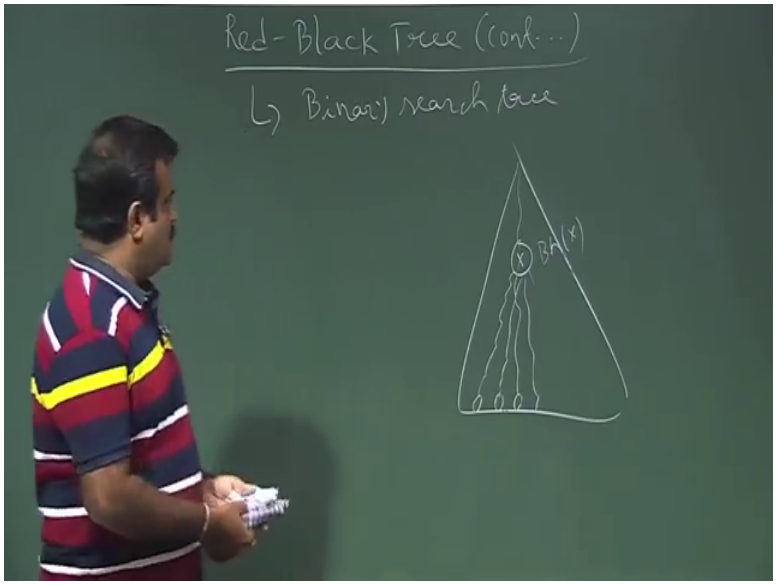
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture - 28
Red Black Tree (Cont.)

So we are talking about balance binary search tree and we have seen the red black tree one of the example of balance binary search tree guaranteed balance binary search tree and we have, so basically red black tree today we will talk about the red black tree insertion and the modifying operation in the red black tree how we can insert a node in a red black tree, I mean.

(Refer Slide Time: 00:49)

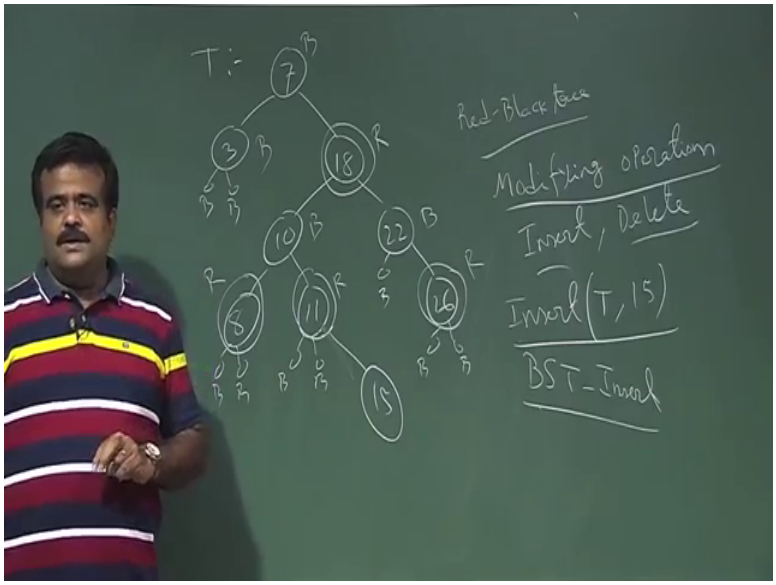


So, basically to recap red black tree is a binary search tree with some properties and there are properties means the every node is colored either red or black and the root is black and the nils nils are basically the new leafs they are black. And another property is if a node is a red the parent must be black and the most important property is the black height. So, if you take a node, suppose this is the tree if you take a node X and if we considered all the path from that node to leaves then if you count the number of black nodes then the number of black nodes are same and that is basically black height of X and that exclude the color of X, if the color of X is black then we will not count that black we not count that. So, excluding the color of excluding the X we just count number

of black node in the path from X to any leaf and that has to be same. So, that is one of the property to be this tree binary search tree to be red black tree.

So, if all this properties satisfied then we have we called our BST or binary search tree is a red black tree and we have proved that in the last class that height of the red black tree is balanced, I mean it is log n if there are n nodes then the height is order of log n. So, today we will discuss some modifying operation in red black tree and those are required for insert a; inserting a node in a red black tree.

(Refer Slide Time: 02:49)



And one of the example we had is like this. So, we have 7 3 18 10 22 8 11 26 and we have the nils the new leafs to make this tree is complete binary tree and then we need to color it. So, this root has to be black and the nils are black, so these are all blacks. This are all black node and then we have to color it. So, this is also we make it black this is black. So, this we make it say if put it like this red and then this node it red, red, and then this is black and then yeah. So, that is it. So, this is the, so if you count the black height of this from this to this node 2. So, if you count from here to this way. So, this is, so these has to be then this has to be red, yeah this red basically. So, if you count the black height of this node I mean this node, so this is from this to this one I two from this to 1 2 yeah, so this has to be red. So, this is a example of red black tree and we can see all the properties are satisfying here.

So, now, we considered some modifying operation in red black tree, modifying operations like we want to insert a node insertion deletion we want to delete a node. So, this set is dynamic set. So, in this set anybody can joint and anybody can leave suppose we have a data base and the every record is having this key and based on this key we make this red black tree and the data base is this set is dynamic. So, anybody can joint any record can joint. So, in that case we have to insert a node any record can get deleted. So, in that case we have to delete a node.

So, what is the advantage of this type of balanced tree? Advantage is suppose we maintain this red black tree for our data base now if you want to search a node. So, searching will just take the (Refer Time: 05:51) high height of this tree because if we search some node. So, we check with the root if it is greater than we will go to the right part of the tree if it is less than we will go to the left part of the tree, so like this we will just make the comparison up to the height of this tree and height is log n. So, the search will take log n time. So, this is the major advantage searching finding minimum maximum successor producer we talk about bit those. So, those will take the height time when height is log n that is the major advantage.

So, let us talk about how we can modify such operation how we can insert a node. So, and deletion we will not cover in this class. So, for our syllabus we have insertion deletion is little more TDS. So, for insertion what we do? We first insert a node in this tree by using the binary search tree insert like we like if you want to insert say 15 here, suppose we want to, so this is our tree now suppose we want to insert, insert in this tree 15. So, what we do? We first do the binary search tree insert. So, how to do the binary search tree insert we will just compare 15 with 7. So, we will go to the right part it is greater than right then again we will compare 15 with 18 we will go to the, so it is less. So, we will go to the left part again 15 with 10 right again 15 is greater than. So, we will put this 15 here now 15 is inserted by the BST insert binary search tree insert. So, we have inserted this new node, now this is the red black tree, so every node has to be colored. So, we want to give the color of this node we want to we have to give the color on this node new node.

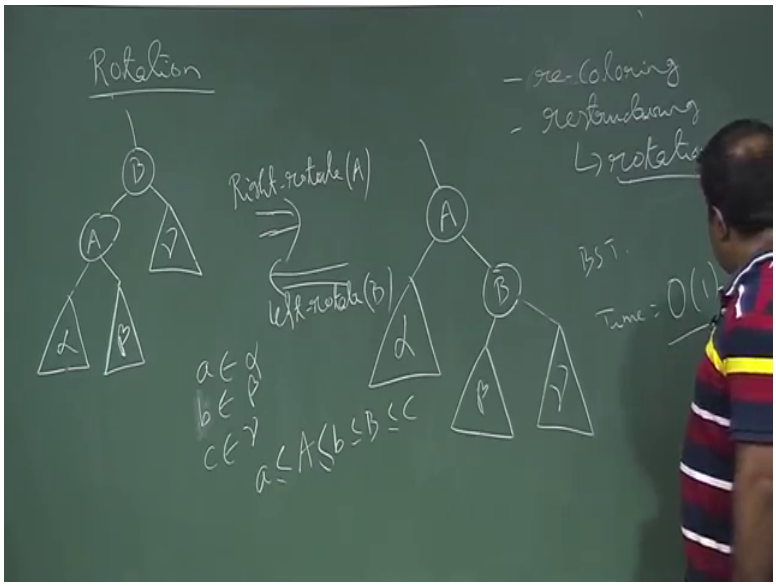
So, which color we should give - we have two option red or black. So, which is your suggestion I means which color we should give on this node. Now the issues is if we give a black color then there is a danger because if we give a black black color and if we

go to a upper node which is having two branch then the black height is hampering I mean then for that node we have the black height is not same in the both side. So, giving black color is that way it is not good idea. So, instead of that if we have to give the if afford to give the red color in that case if the parent is red we give the red color for this new node now if the parent is black of this new inserted a node then there is no problem because we have inserted node which is red. So, every everywhere the black height is preserving. So, only issue may be its parent may be red, now if the parent is black there is no problem still the tree is red black tree.

Now the issue is if the parent is red like in this example. So, so we give this color red now the parent is also red. So, we have a problem over here. So, we need to fix this problem. So, if this is black then we have no issue we can just state away stop here insertion I mean insertion positive stop if this is black because now this got red so we have a trouble here. So, we need to fix that, but we have to give red here because otherwise we will this red black height will be hampering for the upper node which have another branches. So, we give the red color and if we give the red color then there is a problem with one of the property that is if a node is red the parent must be black.

Now if parent is black then we have I mean you are done, but here parent is red so you have to do something. So, to fix that what we can do we can just try to recolor it. So, this is; so how to fix it? So, to fix that we can do this operation like recoloring like we will try to see whether by recoloring we can solve the problem.

(Refer Slide Time: 10:32)



Now, sometimes it is not possible to solve this problem just by recoloring. So, now, in that case what we need to do we need to restructure the tree and restructuring means we need to perform some rotation operation this is called restructuring, restructuring means we need to perform the rotation operation. So, there are basically two types of rotation operation we will perform. So, we will talk about that. So, first, so let us talk about rotation operation. So, rotation means suppose we have a tree like this we have a node B we have a node A and then we have a sub tree rotate here which is denoted by gamma and we have another sub tree alpha we have beta. So, I will come back to this figure let us just what the timing is this.

So, now, this is suppose this is a part of the tree. So, we have other portion. So, this is a node B this is a node A and this is left sub tree rooted at a right sub tree rooted at A and this a right sub tree rooted at A and this are two nodes. Now how to perform rotations? So, this is basically rotation on A. So, we want to have A up B down so that means, this side rotation this is right rotation on A. So, right rotate right rotate on A we can denote. So, this means want A up and we want B to be down and we want this. So, this is the alpha, this is beta and this is gamma.

So, this is basically restructuring the tree. So, this is the rotation operation we are performing. So, we make A up, so the left sub tree with a was alpha still it is with A and now A has a right sub tree beta the node beta now the beta B and now beta beta is the

representative of the any node in the, it was the right sub tree of A now it is a left subtree a now it a left sub tree of B and gamma is still the any node in the right sub tree of B. Now if you do this still we are preserving the binary search tree property so that means, if we take A belongs to alpha B belongs to beta and C belongs to gamma then still we have A less than equal to A less than equal to B less than equal to beta less than equal to C sorry this is B. So, this is the binary search tree.

So, any node alpha is a representative. So, any node in alpha must be a which is less than less than A any node beta must be less than greater than A, but less than B still it is preserving here also. So, this is still a BST and the color also will be same, so this is the rotation operation we are performing. And now how much time it will take to perform this, and the other way round suppose we have this structure we want to make B of A down then this will be the left rotation left rotate on B. So, we want B up A down. So, if this is our potion of the tree and we want to make like this. So, we have to perform left rotate on B. So, this is the rotation operation.

So, how much time it will take to perform this rotation? If basically what we are doing we are changing some pointer. So, pointer means we, so parent of parent of B is not parent of A and the left child of right child of A is now B. So, just we are changing three four pointers. So, it will take time is basically order of 1, order of 1 to perform this rotation operation because we have just changing the few pointers. So, this is the rotation operation and we may need to perform this rotation operation in order to perform this modifying operation like insert deletion. Let us go back to the insertion 15 in that sub tree, in the tree so let us just draw this picture again.

(Refer Slide Time: 16:12)

So, this is 7 3 18 10 22 8 11 26 and we have inserted 15 so we have inserted 15 and we have the nils over here, the nils over here and let us give the color also this is black this is black, this is black, this is black, now this is red, this is red and this is also red now we just make this is to be red.

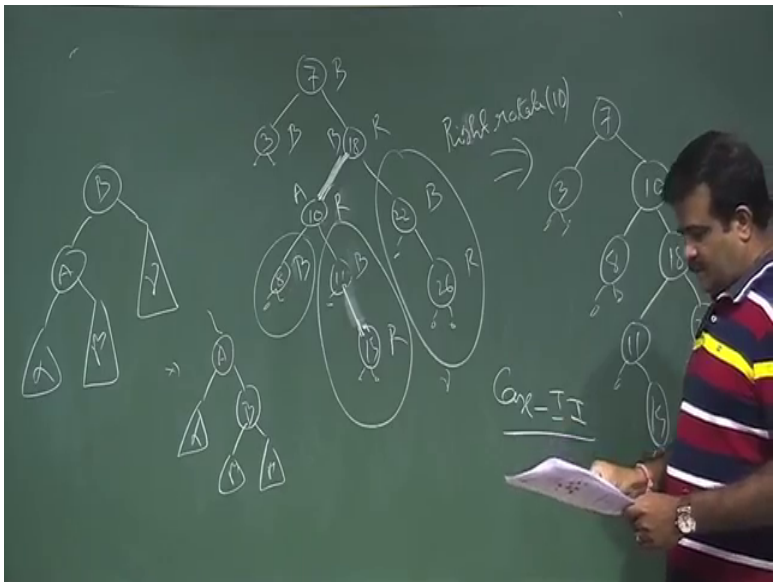
So, now, this is a now; now there is a problem because if a node is red the parent has to be black, but here the node is red parent is also red. So, we have to fix that now we first we try to fix using the recoloring. So, now, this is what is refer as case I, this situation case I is basically, if we look at here parent is red. Now parent that grandfather or grandmother of this node is black and this is the aunty or uncle aunty or uncle is red. So, this situation is called case I. So, if the aunty or uncle is same color with the parent and grandfather or grandmother is black then what we do we just change the color of this. So, we will make this to this is black and this is red and we have this is red, this is black this is red now we solve this problem. So, this is called case I.

Now we solve this problem, but we still have problem over here. So, it propagate this problem up, but this is which is refer as case I. So, case one means like this. So, if we have a situation like this. So, if we have a node A and then we have a node B, there is some tree hanging over here, some tree hanging over here and we have a node D some tree hanging over here some tree hanging over here. So, if this is X, now if this is, so this is red and this is red, this is red, this is this side we have taking example of other side and

this is black. Now this is refer at case one case one means the parent is red aunty or uncle is red, but the grandparent is black. So, in that case what we do - we change the color of these two we make it red and we make it black now it will solve the problem locally, but problem is if this still red, then we have a problem the situation like here. So, problem is still remain because this node is red and the parent is also red.

So, now, we will see whether still we are in case one or not. So, here parent is red grandparent is black, but the aunty or uncle is not red aunty or uncle is different color black. So, this is not case I. So, we have to do we have to perform case II. So, case II means will just perform the we need to perform the rotation operation so we want to, so we need to perform rotation operation. So, we will just do we want to make this up and 18 down. So, this is the right rotate on A. So, this is case II.

(Refer Slide Time: 20:23)



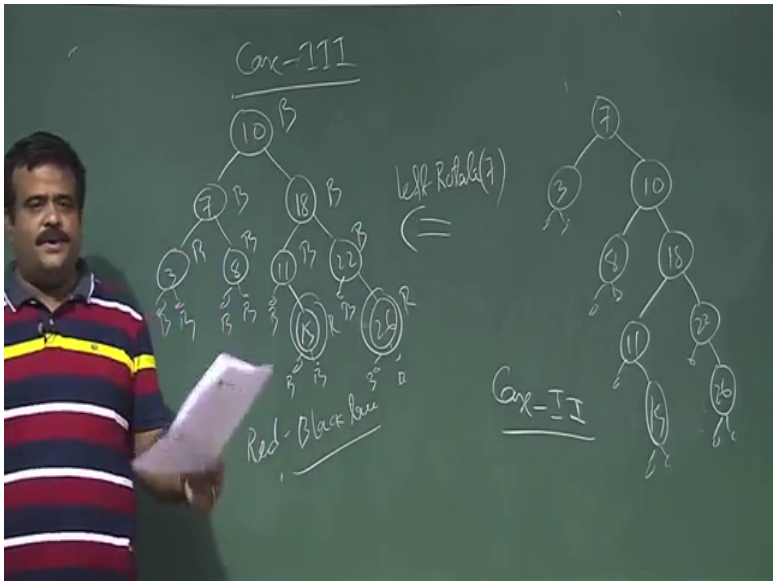
So, what we do, this is A node this is B node if we recall. So, if we recall this is B node this is A node and we have a sub tree rotate over here alpha beta and here we have gamma then if we perform this operation. So, this is basically A node, so we have to will just do what we will just do the right rotate on right rotate on 11, its 10 sorry. So, we want to make 10 up this down. So, in that case this is alpha, this is beta and this is gamma. So, after this operations situation will be like this A will be up, B will be down, and alpha will be still hanging over here and beta will be here and gamma will be here like this. So, let us write this after this rotation. So, 7 will be here. So, this will be

unchanged now this we want 10 and here we have 18 come down and here alpha will be remain same so 8 and then we have this leaf nils and from this 18 we have this is basically beta. So, beta is this 11 and 15 and we have still gamma is with 18, so 22 and 26 and we have the nils over there.

So, this is the situation if you rotate this on 11 (Refer Time: 22:26) 10. So, if you want to make ten up 18 down. So, it will be like this. Now this is called case II. Now if you have to do case II then we have to perform the case III. So, case III means now we want to make this down this up. So, we want to make left rotate on 7 to make 7 down 10 up, so that we will perform now.

So, now, we perform case III, case III means which should be followed by the case II.

(Refer Slide Time: 23:09)



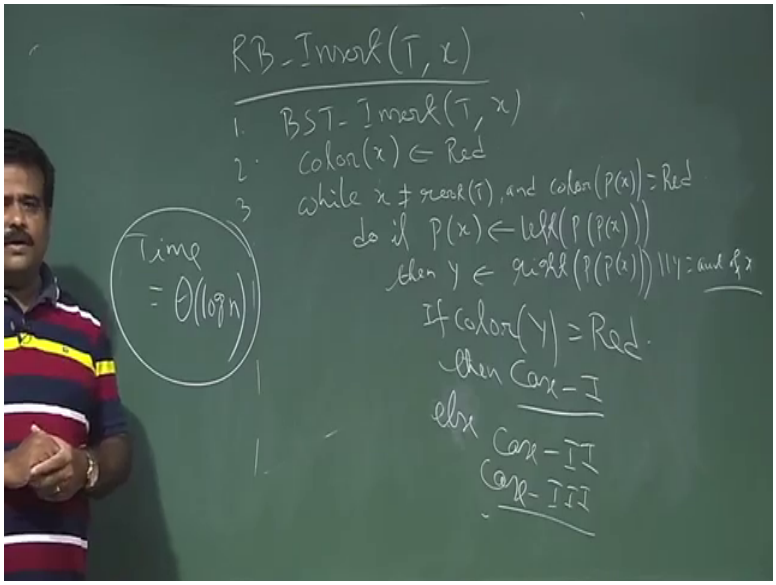
So, here we want to perform left rotate on 7. So, if you do the left rotate on 7. So, 7 will come down 10 will come up, so the situation will be like this. So, 10 will come up 7 will be down and then it is 18 and 3 8, just the left rotate we have seen the operation of left rotate 22 and then we have 15 over here, 26 and this are the nils basically.

Now we do the coloring this is black, the nils are all blacks and so this sorry this is black nils are all black. So, we can make this black black black, this is also black black black and these two we make red we make this two red this recoloring we can do at any time. So, this is basically, this is basically a red black tree if you can just and this is basically

operation is case III, if you have to perform case II then case for case III will be followed by the case II. So, this is a red black tree.

So, this is the insertion operation on a red black tree. So, if we have a red black tree if we want to insert a node then we have to perform this operations. So, let us just write the pseudo code for this, let us just quickly write the pseudo code for this.

(Refer Slide Time: 25:26)



So, red black tree insert, we have tree and we want to insert a node x in the example it was 15 the key values is 15. So, first of all we have to do the BST insert then we have to color the node. So, if afford to give the red color. So, now, if we give the red color it may violate only one property the parent is parent is also red. So, if that is happening if the, so while X is not root if it is not the only node and the color of parent, color of parent of x is red then we have a problem then we have to perform this - do if.

Now case one means if the aunty or uncle is red and the grand grandparent is black. So, that is the case I. So, if p x is the if the parent is the left child of the grandparent p p of x then y is the other child right child; that means, y is the aunty or uncle of x, y is the aunty or uncle of x. And now if the color of color of y is red then it is case I and then if its then we perform the else if it is not if it is not case I else we have to do case II and followed by case III anyway this is the pseudo code for this insertion. So, how long time it will take. So, basically time complexity is order of log n because we have just inserting a node and we have just performing the operation and we have maybe we having doing

rotation this recoloring will take constant time and rotation also take constant time. So, we have just rotating and we have just going to the up of the tree. So, basically we are travel we are going to the height of the tree and this is already balance tree, so height is log n. So, it is this insertion will take log n time.

So, similarly deletion also if you want to delete a node from a red black tree, so we deletion is not we are covering in this course, but if you are interest you can read the text book deletion is little TDS we have to find the success array and then. So, first of all we have to do the BTS deletion binary search tree deletion. So, that is also we will take log n time because that is basically height of the tree. So, red black tree if you want to insert a node then again to make the red black tree we need to spend log n time, if you want to delete a node after deletion again if we want to make this set red black tree it will take log n time, so deletion and every; so deletion and insertion will take log n time.

Thank you.