NPTEL
NPTEL ONLINE CERTIFICATION COURSE

Course Name
Fundamental Algorithms:
Design and Analysis

by
Prof. Sourav Mukhopadhyay
Department of Mathematics
IIT Kharagpur

Lecture 11: Red-Black Tree

(Refer Slide Time: 00:25)

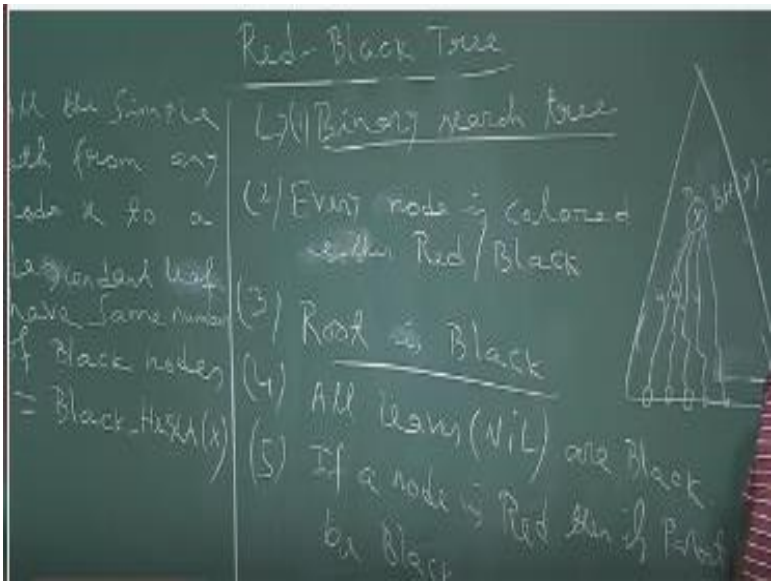
Okay, so we talk about binary search tree, balanced binary search tree and we talk about guaranteed balanced binary search tree. Lastly we have seen that if we have a input we can just do our random binary search and then we can just form the BST, BST insert. Then expected that will be the good tree like, it will be the balance tree, but expected, expected idea is logn.

But here we want guarantee, we want that it should be always logn, not the random permutations or anything it should be always logn, we want to have a balanced binary, guaranteed balanced

binary search tree always. So there are few search tree exist, AVL tree, B tree, then 2-3 tree, then 2-3-4 tree, and Red-Black tree.

So we will talk about Red-Black tree which is a guaranteed balanced tree, balanced binary search tree. Okay, so let us talk about Red-Black tree, first of all it is a binary search tree.

(Refer Slide Time: 02:17)



So that means it has the binary search property so when this is our tree, so we put again node X, then we look at the, this is the key value of that node, we will look at the left sub tree and right sub tree.

So all the key value in the left sub tree might be less then X, all the key value must be greater than X. So it is the key value of this node. So this is what is the binary search property. Now we have few more properties to have a binary Red-Black tree and every node is colored, every node is colored by two color either red or black.

So in a node we have one node data information which is used to indicate whether this node is colored by red or black, okay. So and the route must be, route is black, route is black, this is

another property, another property is all the Nils, Nils means we will introduce the Nils as a lead node, so all the lead nodes which are basically be defined as Nils we will talk about sorted Nils are black.

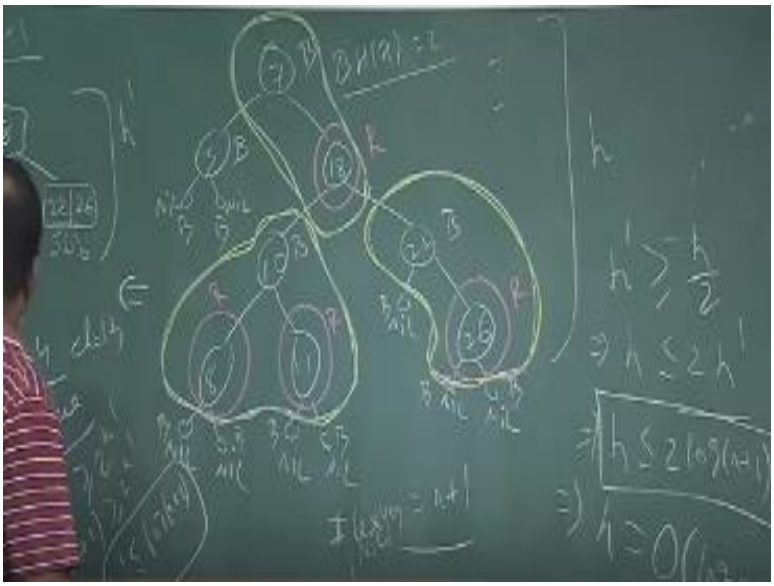
Okay, and this is another property and another property is if a node is red then the parent must be black. If a node is red, then its parent must be black, is black, okay. Now suppose we have a node X which is red then its parent, this has to be black, this is the, this is P(x) parent of X, this is one of the property. And the sixth which tells, last property is very important property which is, which will give us the black height of a node.

So it is telling us all the simple path from any node X to A to a descendent leaf have same number of black nodes, number of black nodes. And that is defined as black height of X and this count is excluding X. So that means if you take a node X we got here okay, now we consider all the path from a single path to its leaf okay. So these are the path which leaf is there.

Now we count each path we count the number of black nodes, so this each path is having same number of black nodes and that is defined as black height of this node X. If this path having four black nodes this path also having four black nodes, four, four like this. And this four is excluding X, so if X is black we will not count this X, that means we are not considering the color of X, excluding X we count the number of nodes from X to its descendent leaf.

And that is the black height of this node and that is, this is the very important property for this Red-Black tree. So if a tree is having this, satisfy all the properties then we call that tree is a Red-Black tree and we will show that it will guarantee height is logn if we have this property in a tree. So let us take an example of a Red-Black tree.

(Refer Slide Time: 07:23)



Okay so suppose we have this tree, root is 7 and 3, 18, then this is 10, 22, 8, 11, and we have 26. Now let us introduce the Nils, leap nodes we what to make complete binary trees so we want to introduce leap nodes, these are the Nils, these are basically new leaps, these are called Nils, this we are introducing these are one child is there so this is to make the trees complete binary tree, okay.

Okay, now first of all this a binary search tree, if we check any element this left sub tree is less than 18, right sub tree all the key values greater than 18, okay, now we need to give the color, so can you suggest some color for this? So now root must be so all the Nils must be black so they are all black, they are all black and the root is also black okay, now can you suggest some color for this, suppose we color this as a red and we color this as a red, this as a red and we color this as a red, okay.

So we have to check whether this is satisfying all the properties or not so every nodes has been colored so this is also black, this is black, this is black so we color every node, so now, so now the root is black and the Nils are all black so that is one of the property, now if we consider a node which is red we have to check whether parents is black or not so this node parent is black,

this node parent is black, this node so that property is also satisfied so only thing we need to check the black height. Now we can check what is the black height of this node, black height of this node is one, black height of this node is one, what is the black height of this node, if you take any path from this two is descendent leaf.

Then black height of this node is basically one, so what is the black height of this node it if you take this path? One black node, one two if you take this path one, two if you take this path one, two so black of this node is two, now what is the black height of this node, now if you take this path one, two so what is the black height of this node, 7 this is black height of 7 is basically this by 2 now if you take this for A so this we have to exclude the color of this.

So how many black nodes one, two even this way, even if we just count so that height of this node is two, so it is satisfying the this black height property, so every node if you take the simple platform diagonal to the descendent leaf we have the same number of black nodes and that is the black height. So good news this is a red, black tree, so now we want to prove that how this is height is guaranteed log n so we have to prove the theorem, we want to prove a theorem what is telling us for a any red black red tree, a red black tree with n nodes, suppose there are n nodes, n nodes has height $h \leq 2 \log n + 1$ so that means h is O(log n), so this is balanced.

So how to prove that, so to prove this what we do we want to remove the red color, so how we want to remove the red color, if you just want to make it black color then there will be a problem of the black height so we cannot just change the color of red to black, so what we do we just we know if a node is red its parent must be black, so what we do we merge the red node with this parent, we take a red node, we know the heights parent is black so we will merge it, so we will merge all the red nodes which is parent.

So this is red node, we are going to merge it with this, okay this is red node, we are going to merge it with this, these two are red nodes and this is the parent so we are going to merge it with this, okay. So this is the merging we will do so this is basically you are merging this, so these way we want to remove the red color, so now we do not want any red color in the node, in the tree, so if you do this then what will be the new tree, so this tree will convert it into this tree

where this is 7, 18 and we have 3 over here and this is the nil and here we have, so these are merged, so how may nodes?

So 10, 8, 11 and there are, this is basically 22, 26 and we have how many Nils, three Nils. Okay, so this is the, this is the new tree after removing the black color so basically, so this is, this is, so every node in this tree has how many child? Every node in this tree, node has either two child, or three child, or four child. This is called two, three, four tree, this is a very good, very balanced tree you can see, okay. Now if we look at the height of this tree, if the height of this tree is h and if the height, so this is the height of this tree is h prime. So can you have a relationship between h and h prime?

Which mean that is more and which so h, so h prime is, so can you have a relationship between h and h prime, so can you write this? h prime is greater than equal to h/2, why? See I mean we are only, if you take a path which is getting of this maximum depth that is the height you take that path, in that path we have what is the maximum color node, red or black? That black color is more because if you take a red color its parents must be black.

So if you take a path the number of red color is less than number of black color, so that means we are merging only black, red color, so that means the height of these will be greater than half of the height of this node because half of the height of this tree because otherwise the rate color will be more in a path which is not possible, because if a node is red then it must be black. So if you take any path the number of red nodes is less.

So that means so this property satisfies, okay good. So each prime so height of this tree is greater than equal to so h is less than equal to 2h prime, okay. Now how many leaves are there? Number of leafs for this node n+1 and we have same number of leaves, leaves are basically leaves ,we have same number of leaves in this node also because we are not changing, the we are not merging the leaves because leaves are already, leaves are not red, we are only merging the red nodes.

So leaves are not red leaves are black, so number of leaves is same so number of leaves here is same as the number of leaves in this tree which is n+1, okay. So if the number of leaves is n+1 and we know that is each of this node have child minimum to maximum four so can you write this, so n+1 this number of leaves must be must be greater than equal to $2^{h'}$ because each child have minimum two, each node has minimum two child.

So this is true, now this will give us so $(n+1) \geq 2h'$ so $h' \leq \log(n+1)$ so this we will use here, so that means this imply 2h" so h' is log (n+1). So this means h is O(log n) so balanced, so red black tree is balanced tree guaranteed the node under is nothing so it is guaranteed, it is a balance binary search tree, if we have having all the properties in the tree, okay. So now it is a good news since it is a balanced.

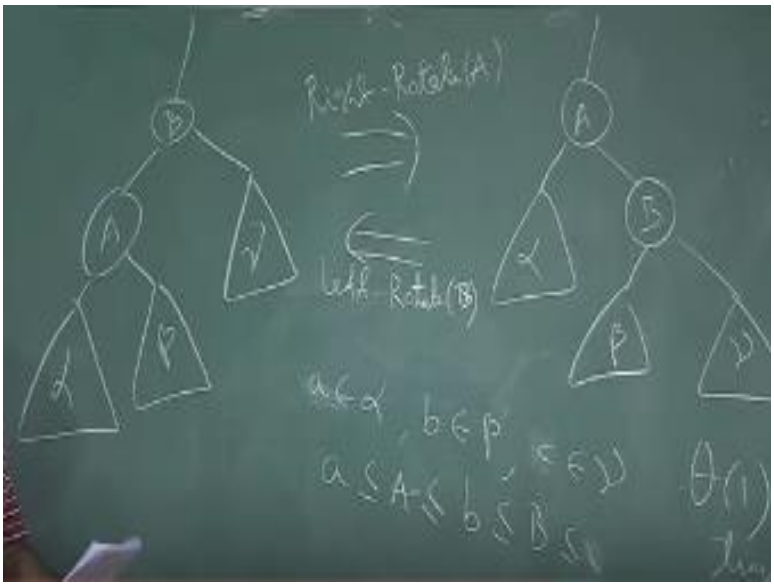
So we can find the minimum if we have a some numbers and if we can store in a red black tree way then we can find the minimum maximum or we can find the successor in a because for every time we have to may be traverse the tree from root to that leaves, height of the tree, so if the height of the tree is log n so those square will be taking log n time, okay. So now we want to see how we can do the modified operation in this red black tree, like insertion. Deletion, we will talk about insertion, deletion is not in our syllabus.

(Refer Slide Time: 19:02)

So how we can insert a node in a red black tree, insert, we want to insert a node whose key value is x in a red black tree. So for that we have to do some modifying operation in a red black tree, so first operation may be we need to change the color, color change, this is very simple operation and it will take constant time, we will go to that node, we will, if we if our convention is if the node is red.

One bit of information we need there if the node is red we will put it 1 if the node is black we will put it 0 like this, okay. And the second operation is which is basically re-structuring, I mean you need to really do something on the re-structuring which is called rotation, rotation operation which may required, only coloring will not help us to make the tree again a red black tree. So we will define the rotation operation.

(Refer Slide Time: 20:21)



So suppose we have tree like this we have a node B over here, we have a node A over here and we have the upper part of the tree and suppose this is denoting by α sub tree, I will pay just to representation this is β and we have a tree hanging over here is γ, okay. So this is B node, A node and this is the α so that means all the α is the just a representation symbol, basically we denote any keys over here, key values α. So $\alpha < A$, $\beta > B$ so this is binary search heap property, okay.

So we want to make A up, B down, that is call right rotation, right rotate or A, we want to make A up B down so how we can do, so this operation is telling us A up so whatever the parent pointers of B, now it is the parent pointer of A, now B will be come down and the tree will be like this α will be hanging here, and this is β and this is γ, so this is the, this is the right we want to make A up B down so this is the right rotation on A.

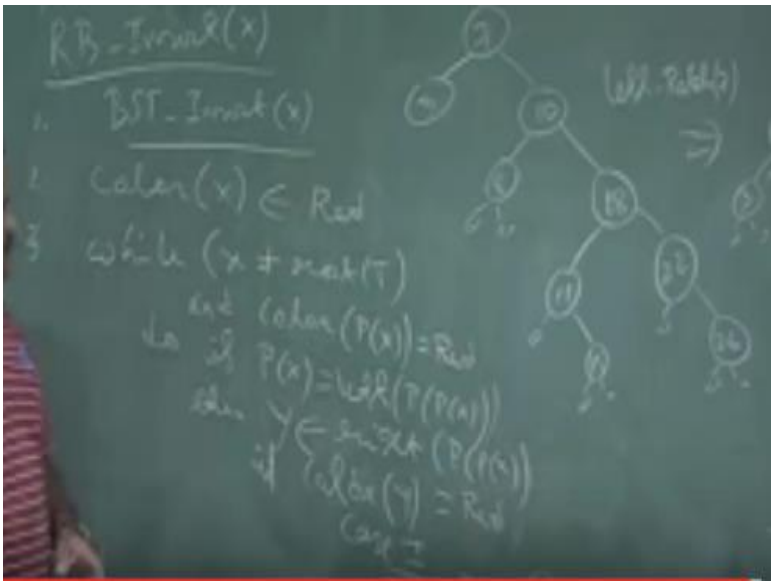
So, so this is also satisfying the red black binary search tree property, we can just if any node in a α, and then we take any node whose key value is b in β, next we take any node in γ and we have this property a is less than A, less than b, less than B less than c, okay, so it is satisfying the binary search tree property. So how much time we are spending here for this rotation, we are just changing few pointers because the parent pointers of B, now the parent pointers of A, now this A

now the B is having left child of B was A, now the left child of B is now this sub tree, so we are just changing few pointers, so this is basically constant time, so time for this is basically constant, because it is nothing but changing some pointers, okay.

Now suppose we have this node and we want to make A down B up this is also and the rotation operation, this is called left rotate on B we want to make B up A down, so other way down left rotate, left rotate on B, okay. So we want to make B up A down so similarly this is the input then this will be the output under this rotation operation and that is also a constant time operation because we are just going to change few pointers, okay. So we want to use this, we mean it to this is the restructuring, we are restructuring the tree so we need to use this for our insert operation.

So let us talk about the insertion, so these are two modified operation, one is recoloring and the second one is this rotation, with this help of this two we want to insert a node.

(Refer Slide Time: 24:20)



Okay, so suppose let us go back to our this tree, so18, 10, I write it here so that we can do the operation, so this is 7, this is 3, this is 18, this is 10, this is 22,8,11 and this is 26 and we have the

nils, we have the nils, okay. Now suppose we want to insert x, insert and node x whose key value is x suppose x is 15 we want to insert a node whose key value is 15.

So what we do we first do a binary search tree insert BST insert, so we put a BST insert it will go it will check with this, it is 15 is greater than so it will come here, 15 is less than it will check with this greater than so it will inserted here 15, now we will put a Nils. Now we need to give a color so what was the color for this node so this was red, this, this two are red, this is the, this is red all nodes are black node, okay.

So now, sorry 15 is recently inserted so which color we should give on 15, if we give black color than there is a problem with the black height, so step side we can give it red color, we give the red color on 15 so if we give red color then there is only one property may be highlighted that is the parent must be great, if you notice red parent must be black, so only that property maybe highlighting.

If this parent is black then we are done nothing to be needed to do for that, but here we are the parent is red so we got a problem so how to solve this problem, now we will, we will forward this problem half, okay. Now parent is red this parent now we will check this parent is red we will check the parent, this is aunty or uncle of this, this node because this is the grandparent this is the aunty or uncle so we will check the color of this, if the color of this is red then what we do we change the color this is what is call case one will change the color of this so we will make it black and we will make we will make it black this two now black and we will make it red this note okay.

So this local problem has been solved but these will create a global problems though this problem is for data because this parent is also rate now this, this aunty or uncle is not same color so this is called case two so for this what we do we have to perform the only the coloring changing coloring in lot help us.

So we have to perform a cooperation rotational operation so we will perform the right rotate on tale so make 10 half this is down so if you do this so if you remember an so this is our α this is

our β and this is our γ so if you remember that so this is 7 so this is 3 now 10 will come here 10 is basically our so this will come down 18 now it is having 2 part so this part and this and this 10 will be having just this 8 and this will be hanging 11 and this will be hanging 11 and this 15 just remember α β γ and this will be 22 and 26 like this.

So now so this is case two now if we apply case two we have to apply case three so case three is basically now we have to give a right total on 7 so you have to make this up so left rotate on 7 so we want to put this half 10 half 7 down so similarly α β γ it can have so 7 down and then we have 18 over here and then with 7 you have this 38 and then 18 we have 11 and then 22 then here you have 15 and we have 26 and all the means are here okay so now we need to give the color so which color we will give we will just gibe so these has to be black.

And we will make this is black this also black so this is black this is black so you can just return the original check the original color but these two will put it rate okay so, so if we have to get this is called case two this right case three sorry so if we have to apply case two then it will be then this must followed by case three and then after case three we will do some color change to make a red black tree so this is a red black so this is the insertion of a known so if you want to write the code syndicate for so red black tree insert of a so what we are doing we are just doing the BST insert.

Insert of x and then we are putting the color of this so color of this we are putting as rate so that may let one property now we are checking while if this is just a root then we are not doing anything if this is not a root and if the color of parent is then we got a problem of the color of parent is red then you have problem if it is black then there is no issue if it is rate then what we do to if, if the parent is the left child of the parent of parent then we check the color of is aunty or uncle then Y is the right child.

Of the parent of parent this is basically uncle or aunty of X and if uncle or aunty if red if color of y is red then this is case one and if case one if again the same problem is there then we will apply case two and then followed by case three okay so this is the she cot for this so how much time we are spending to insert basically every time it is basically we can go up to the root so it was a

weight black tree now we have inserted so it is basically log n time reinsert in note maximum we can we need to go for up to the root level so this is log n time height of the tree, thank you.