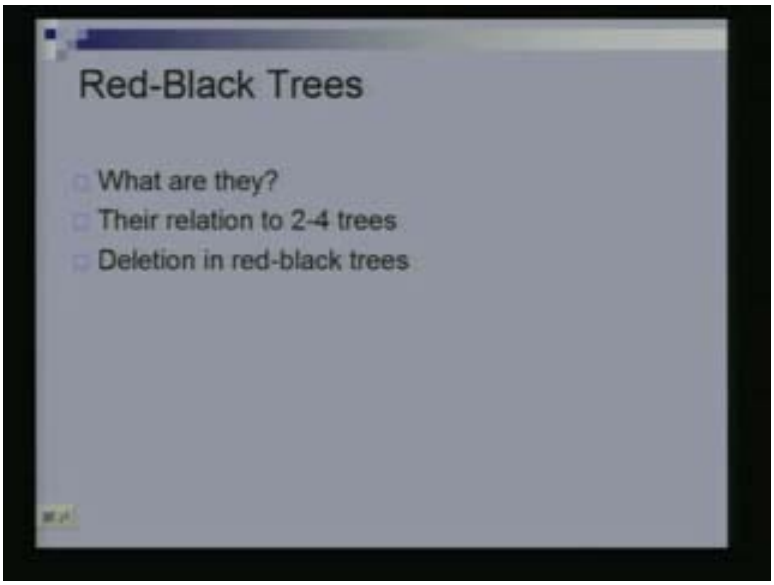
Data Structures and Algorithms
Dr. Naveen Garg
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Lecture – 14
Red Black Trees

In today's class we are going to talking about red black trees. We are going to spend some time discussing about red black trees, what is the relation to 2-4 trees and what will discuss next and then we will see the process of deletion in red black trees. It is going to be slightly more involved process so we will be spending fair bit of class on deletion, in the next we are going to handle insertion in red black trees.

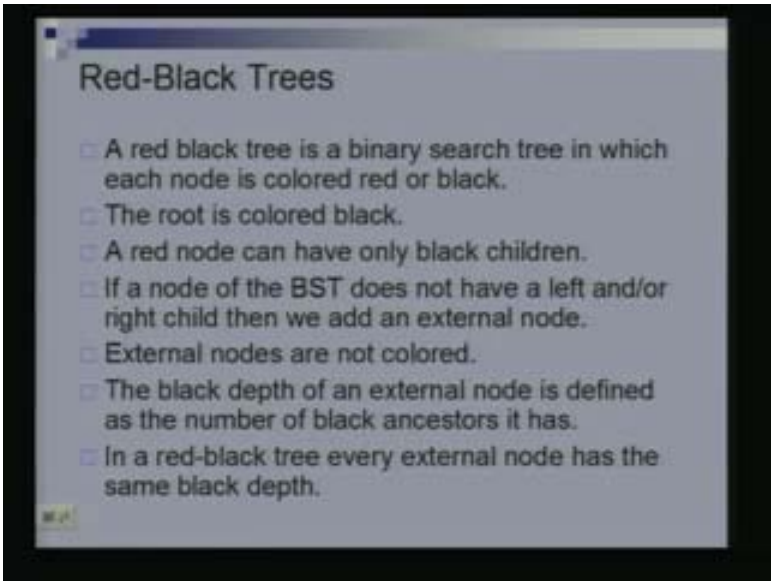
(Refer Slide Time: 01:17)



So what is a red black tree? So red black tree first it's a binary search tree. So we take the node of the binary search tree and we color them red and black and then it becomes the red black tree subject to certain properties. So first the root has to be colored black. That is important. A red node can have only black children that is another property that a red black tree has to have.

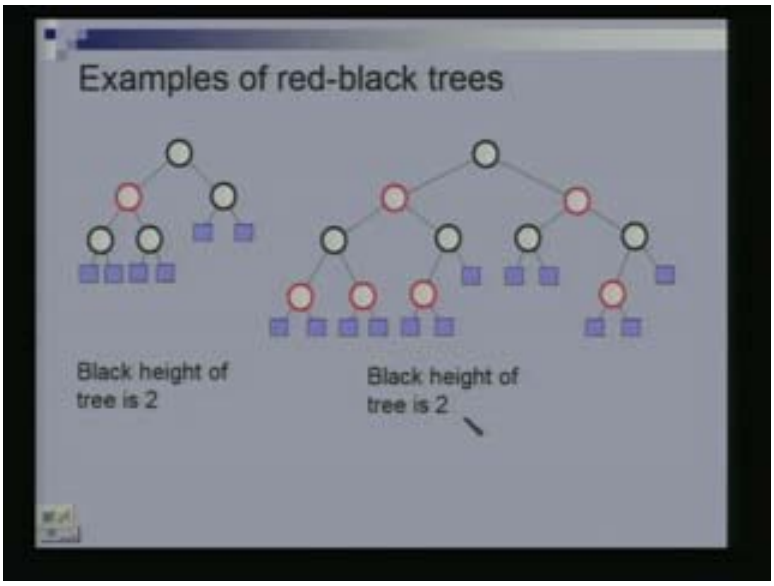
If there is in a red node then its children have to be black. Now one thing we are going to do is in our binary search tree if a node does not have left or a right child, so we will create the left or a right child by creating what we will call an external node. And I will show you what I mean by that. We will create an additional left or right child, if it's a leaf node of the binary search tree then it does not have either a left child or a right child. So in which case we will put in a two external nodes below that leaf node and make it the left and the right child.

(Refer Slide Time: 01:40)



So external nodes we are not going to color them. So it's only the non-external node, original nodes of the binary search tree that we are going to color red or black. We define the black depth of an external node as the number of black, its number of black ancestor's. So what that means is we are going to take an external node and we are going to walk up the external node towards the root and the number of black nodes we encounter is the black depth of this external node. And the one key property of the red black tree is going to be that every external node has the same black depth and that we will also refer to as the black height of this tree. So lots of definition but we will see what this means, a couple of examples.

(Refer Slide Time: 06:42)



So this is two examples of red black trees. So let us look at the one here. So what do we have? This is the binary search tree on 5 nodes I have colored the nodes black and red. The root is black, a red node has only black children, black node can have black or red children. But a red node can have only black children. As you can see this node does not have any left or right child. This does not have any left right child; this does not have any left or right child. So I am going to add them (Refer Slide Time: 04:20). These I will call external nodes, they are not colored. So all the external nodes in this class today will be shown with the square boxes. Now what is the black depth of this external node? 1, 2. It is the 2 ancestors which are black, similarly the black depth of this are external node is 2, of this is 2, of this is 2, of this is 2 and of this is also 2. So this is a red black tree.

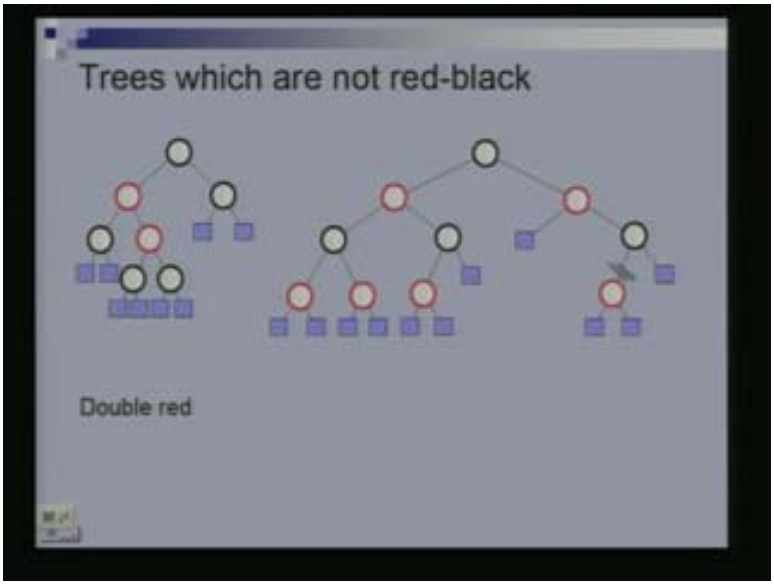
So we will add this external node also to node which has only one child. So let us look at this example and that will be clearer. So once again the root is black, a red node has only black children. That is the key, the red node can have only black children. Black node can have red children or black children. Now we are going to add the external node so as you can this node has only one child. So we will need to add an external node here also. Similarly it will need to add 2 external nodes here, one external node here and of course these will take 2 external nodes and so on. So these are the external nodes we will end of handing. [Refer Slide Time: 05:43] (Student Conversation). A black can have black children but red cannot have red children, it's not symmetric.

[Student Conversation: you going to take first case, staff: ya, Student: it has black and both the children left and right to be red right child is if we place the external node and we count the black height so it will be one Staff: red so then it will not be a red black tree that's what you say.]

So if I had colored this red then this would not be a red black tree. The black height of this guy is 2. What is the black height of this guy, of this tree? 2, I will take any external node and look at how many ancestors are black and we will see that its 2 ancestors which are black. And this is true for every external node; if I look at this it also has two black ancestors, this also two black ancestor and so on. So this also has a black height of tree. So what are the key things we have to ensure? First that no red node has a red child, second root is black, third black height of the every external node is the same. Black height or depth I sometimes use the term depth to say how far it is from the top. So black depth of the every external node is the same, the black depth that's what we will call the black height of the tree, depth of the node and height of the tree.

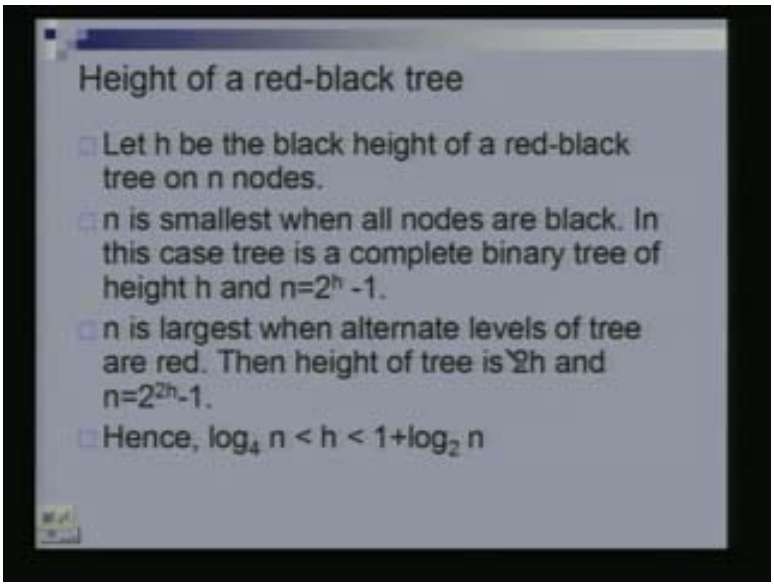
And these are the two examples of trees which are not red black (Refer Slide Time: 07:24). Why this is not red black? This red child, this red node has a red child. So this is not, so even I have put in the external node like this, this will not be a red black tree. Why is this not a red black tree? So the problem here is what we call a double red problem. We will use the term quiet often, double red as in two reds occurring consecutively one after the other.

(Refer Slide Time: 07:55)



Here I add the external nodes, the black height of this node is, the black depth of this node is 2, of this node 2, of all of these is 2. This is also 2, this is one, this is culprit, for this reason this is not a red black tree. So black height is not uniform, black depth is not uniform. So let's look at what is the height of the red black tree is going to be? So let's say h is the black height of a red black tree on n nodes. What does it mean?

(Refer Slide Time: 08:23)



If I take any external node and I will look at, count its number of ancestor that it has exactly h black ancestor. When will the number of nodes in this tree, suppose I just tell you that this is the tree of black height h, when will the number of nodes in the tree be

small? When the smallest? When everything is black, you don't want any red. You want as few nodes as possible. So why include your red nodes and increase the number of nodes. So let's have everything black, if you have everything black then this becomes a complete binary tree of height h or my red black tree why because all the external nodes have to be at the same level now. If they have to be at the same level all the external nodes then it will become a complete binary tree of height h. If I have a complete binary tree of height h then the number of nodes in the tree is 2 to the h-1. So the smallest number of nodes possible in a red black tree of black height h is 2 to the h-1. When is the number of nodes the largest? When every black node has 2 red children, but red node cannot have any red children. So then it will have to have black children which will have to have red children and so on. We will have these alternate layers of black and red. So we start with the black then we have red layer then we have black layer then red layer and so on.

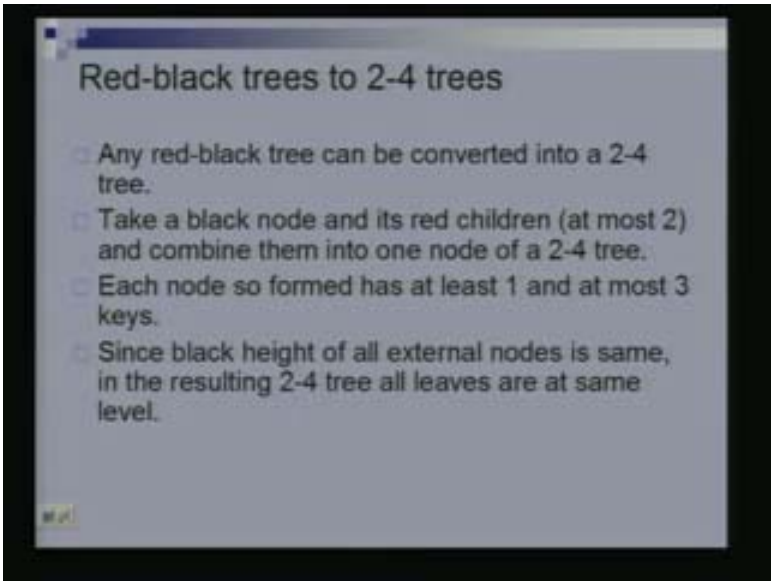
If the number of black layer is h then the number of red layer is also h. So the total height of the tree becomes 2h and so number of nodes in the tree becomes 2 to the 2 h-1 which is about 4 to the h-1. So this implies that h lies between log of n to the base 4 and log of n to the base 2, just +1and -1you can figure out. So if I give you a red black tree on n nodes, so I just turn thing around then if I give you red black tree on n nodes then its height is at least log n to the base 4 and at most log n to the base 2. We have seen exactly of the same property for 2-4 trees. We said its minimum height is log n to the base 4 and maximum height is log n to the base 2. The same kind of the thing is happening here.

So now you can immediately see that if I give you red black tree then to search in the tree, will take how much time now? Log n, why because first remember it is a binary search tree. So it has the search property, so I can do the search in this regular way that is start from the root, compare key with the root, go left or right and so on. So I can do my search and the time taken for that search is just the height of the tree and the height is no more than log n base 2. So that's the time I take.

Now let's look at the correspondence between red black trees and 2-4 trees. So in particular I am going to say that given any red black tree, I can convert it into a 2-4 tree. So what are we going to do? We are going to take a black node, that black node, how many red children will it have? It's a binary tree. So it might have both its children could be black, one child could be red, the other could be black or both could be red, both could be black, both could be red. One could be red, one could be black or you could have external node also, one could be external node but in any case the number of red nodes cannot exceed 2. It will be 0, 1 or 2. So we are going to take this black node and its red children and combine them in to one node. How many keys there will become in this one node? At most 3, when this node had 2 red children. Let's see.

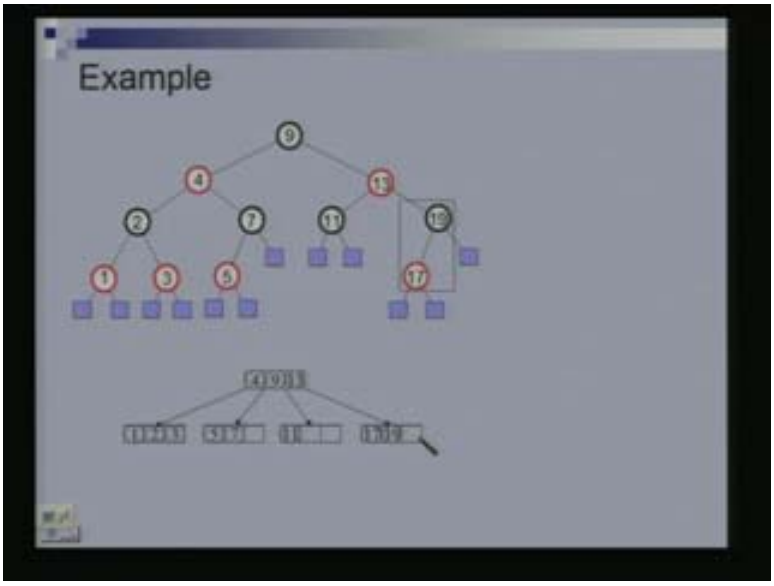
So what I am saying? I am saying, we have a black node and we look at its red children. Suppose it had 2 red children so I am going to combine it in to one node, all these 3 I am going to combine in to one node. So this one node will have 3 children. Suppose it had only one red child and then it would be 2 keys in here and if it had no red child then there will be only one key here.

(Refer Slide Time: 12:00)



So each node so formed has at least one and at most 3 keys. Furthermore black height of all the external node is the same, so in the resulting 2-4 tree all leaves will be the same, will be at the same level and this might has not too make sense. What are we saying? Black height of all external nodes is the same. So if I start from the external node and go up the tree, the number of black nodes I encounter would be the same. No matter which leaf I started from? So each of those black nodes is now part of a unique 2-4 node. Because I took a black node and I took its red children and I combine in to one. So each of a black node is a part of a unique node of a 2-4 tree. So as consequence when I start from a leaf and I go up to the root I encounter the same number of nodes in a 2-4 tree which means that all the leafs are in the same level of the 2-4 tree.

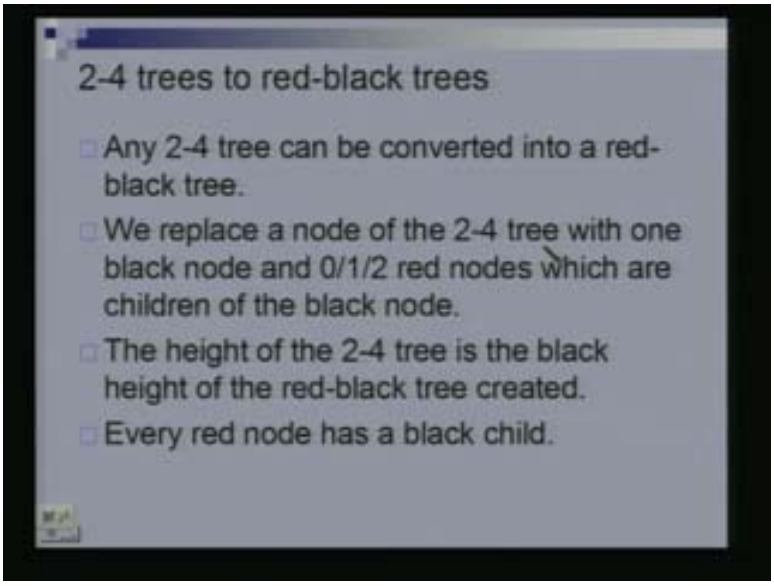
(Refer Slide Time: 17:23)



We will show an example with which this will become clear. So this is my (Refer Slide Time: 15:36) red black tree I am going to convert it in to a 2-4 tree. So I start with the root node, I will look at its red children it has two red children. All of these will combine in to single node of my 2-4 trees. Now I am going to look at another black node which is this, look at its red children it has again 2 red children so all of this combine in to one. And they combine in to one like this (Refer Slide Time: 16:16). So as you can see when these combine in to one, these 4 links that I am going out of this combination will be the 4 links going out of this node. When I took this entire thing let me go up, there are four, 1, 2, 3, 4 children of this (Refer Slide Time: 16:47) combined structure. They would be the 4 children of this node. So this is the first one, then I look at this. How many red children does it have? Only one, so they would combine in to one, so I get another node which is 5, 7. This becomes a second child, as you can see this is more than 4 so this will be more than 4 here.

And what will be my third child? It would be just this black node, it has no red children. So I create a node with only 11 in it and the fourth one 19 and 17, so I create another node with just 17 and 19 in it. So this is the 2-4 tree I get. Now you can see the black height was 2, it was the same for every external that was the black tree and I get 2-4 tree of height 2. So I am giving a proof by example but this should be clear why I will get the 2-4 tree? So the property of a 2-4 tree, one critical property of the 2-4 tree is that all leaves are at the same level. Just saying that every node has 2, 3 or 4 children doesn't make it a 2-4 tree. All leaves have to be at the same level and that is in ensured because of the fact that the black depth of every external node is the same. So this is how we can obtain a 2-4 tree from a red black tree.

(Refer Slide Time: 18:29)

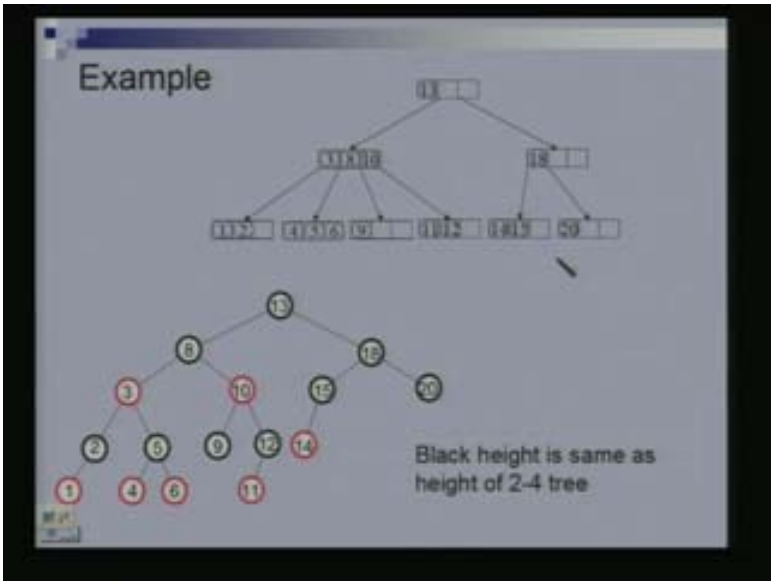


We can also go the other way round, that is given a 2-4 tree you can make a red black tree. So what are we going to do? We are going to take a node of the 2-4 tree, replace it with 1 black node and then appropriate number of red nodes. What do you mean by appropriate number? If they were 2 keys in this 2-4 tree node then I will have 1 red node. If there were 3 keys then I will have 2 red nodes. If there was only one key then I will have no red nodes and these red nodes will be the children of the black node. So first I put down the black node then I put the appropriate number of red nodes.

Now this ensures that I will not have the double red problem. Why? I will not create one red node as a child of another red node, because what am I doing. I am first taking a node of my 2-4 tree, first putting down a black node and then putting 0, 1 or 2 red children and then when I take the next node of a 2-4 tree which is the child of the previous one. Then once again I will first put down the black node. So I will not have 2 reds consecutively happening that we will see in the example shortly. Furthermore what's going to happen is that for every node of the 2-4 tree I am putting in one black node.

So since all the leaves of the 2-4 tree are in the same level. The black height of the resulting red black tree would be uniform; the black depths of all the leaves would be the same. So let's take an example. Ignore this for now, this should have come later. This is my 2-4 tree (Refer Slide Time: 20:21).

(Refer Slide Time: 21:52)



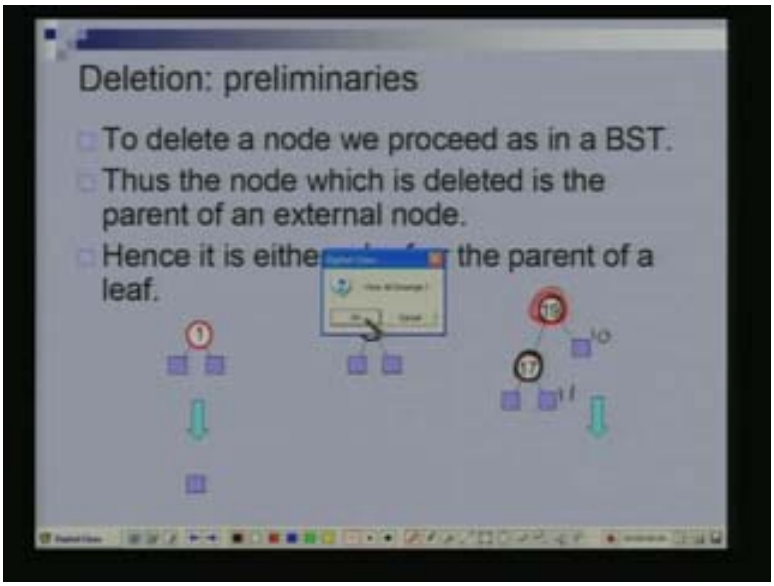
As you can see each node has 1, 2 or 3 keys which mean 2, 3 or 4 children and the height of the tree is 3. Let's see. So there is only one key here, I will just create a black node and nothing else no red nodes. Let me take this one, so I will create 1 black node and it will have 2 children, 2 red children. What should be the key in the black node be? 8, 3 and 10 means 2 children. Then I take this one, 18 just one black node with 18, no red children. Let me take 1, 2 so this would have 1 black node and 1 red node. I have an option; I might put 1 as the black and 2 as the red or the other way round.
So I am going to put 2 as the black and 1 as the red. For 4, 5, 6; 5 will be the black and 4, 6 will be the red, black 9 nothing else. Here 12 black let's say and 11 red, 14, 15 black, 14 red and 20. So now are you convinced that you will not have double red problem? Just by the way we are doing things.

When I take a node and I create some thing in the red black tree I first put down the black node and only then I put down the red node. So at the next level I will first put down the black node. So it will never have two consecutive reds. Furthermore the black height of this red black trees so if I have to draw the external nodes and count their ancestors so if suppose there is an external node here, it has three black ancestors which is as same as the height of the red black tree because of that all of them would have same black depth. If I were to look at this external node which is here then it has black depths of 3. Everyone would have the same because if it is going from here to here then it corresponds to going from here to here because each of those black nodes is one level here. So the black height is same as the height of the 2-4 tree. So you understand what red black trees are and you understand how they are related to 2-4 tree.

Basically there is 1 to 1 correspondence. Given any 2-4 tree I can create a red black tree, given any red black I can create a 2-4 tree. So in fact the operation of insertion and deletion in a red black tree are exactly the same as you do in a 2-4 tree. So we are just going to mimic those operations in this setting and that was one of the major reasons for

doing the 2-4 tree. So we are going to mimic the operation but of course it will require to do it carefully. So I am going to look at the operation of deletion because this is slightly more tricky operation.

(Refer Slide Time: 28:22)



So how do we delete in a binary search tree? The first step of the deletion is the same as the binary search tree which is let we first search for the node, you are in to five other node. If the node is a leaf then you just deleted. If the node is an internal node then you find it successor or predecessor, you swap and then you delete the successor or predecessor. So now the successor or predecessors suppose we are talking about successor so successor is a node which does not have a right child.

If it does not have a right child then that means we would have its right child which is an external node, we put an external node there. So that means the node that I am deleting is always the parent of some external node. If it's a leaf even then its parent of an external node. Otherwise it doesn't have a right child if you were talking of a successor. If it doesn't have a right child we put in an external node. So it's always parent of an external node. So these are the three settings we could have of the node that we are deleting, these two corresponding it to being a leaf.

So either if the node is a leaf either it is a red leaf or black leaf and the third setting is when the node that we are deleting does not have a right child. So it does not have a right child which means right child is an external node, if it does not have a right child but has a left child then this left child has to be red. Why? If it were to be black then black could not be remain consistence. Yes why? Because if I look at these external node verses this external node, [Hindi] you follow what I am saying. If this were black then if this node has 10 black ancestors then this one would have 11 black ancestors. Why? Because all its black ancestors are also ancestor of this guy and this has one more ancestor, this has to be red therefore.

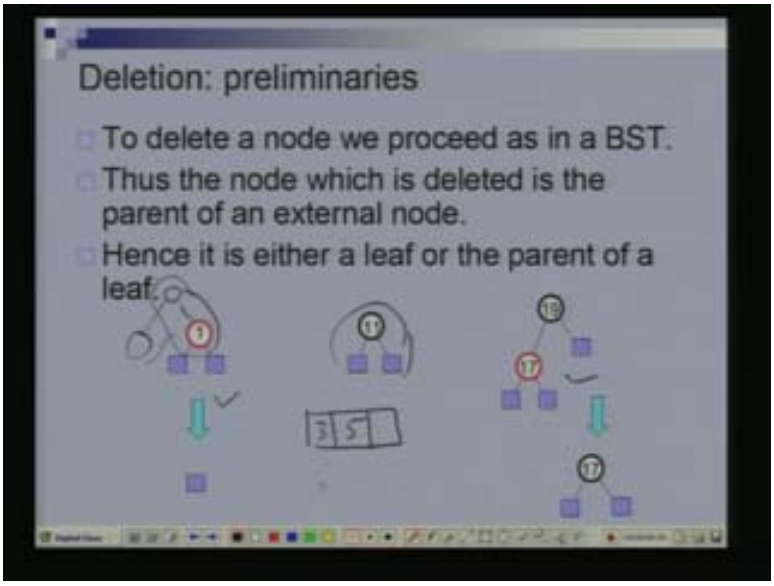
If this is red then it cannot have any red children but it cannot have any black children either because if it had black children then once again we could have a same problem of black height not being the same. So this is the entire structure that we would have. If this is the node that I am trying to delete and it did not have a right child and it was not a leaf either then that means it's right child is an external node, we said that already its left child will be a red node which would be a leaf. So then in this case the node that we are deleting is really parent of a leaf node.

In this case the node we are deleting, in each case we are deleting the node here. In these two cases the node we are deleting or leaf node themselves and this case it's a parent of a leaf node. If 19 is a red node. Can 19 be a red node and 17 be a black node, Black height would not be the same. He is saying suppose this is red and this is black, this is red and this is red, it cannot be. It's a red double red problem. This is red and this is black. [Hindi]. So now this is an easy case to handle. If I am trying to delete this node what should I do? It's a leaf; just delete it, nothing to be done. It cannot create a double red problem, if I delete this. That's what I have done, just delete and just replace this entire thing with this external node. It cannot create a double red problem and it cannot also change the black height of any node because it's a red node after all.

Similarly this case is also easy. What will I do? I am trying to delete this 19. So what should I do? 17 can move there and this entire thing can be removed. So I have not created a double red problem [Hindi] as it was before and this earlier had 10 black ancestors, it still has 10 black ancestors and this node also has same number of black ancestor. So the only tricky case is this one. By the way what do these two cases corresponds to in the case of 2-4 tree? This and this (Refer Slide Time: 30:30). What are we deleting? The key that we are removing is from a leaf always in a 2-4 tree and this corresponds to the case when the key that we are removing is part of node which has at least two keys.

The key that we are removing is in a node which has atleast two keys and that was the very simple case there. If this was my 2-4 tree node and this had keys let's say 3 and 5 and I was trying to remove 5 or 3 its very simple operation nothing needs to be done and this case as well as this case both of them correspond to this (Refer Slide Time: 30:54).

(Refer Slide Time: 31:25)

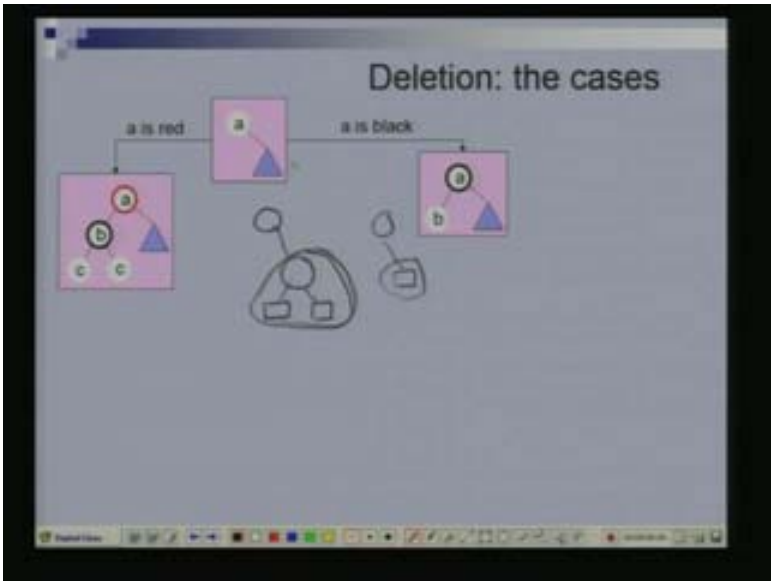


Why does this corresponds to this? This is the node, even here this has to have a black parent and so this is the node then corresponding node in the 2-4 tree and this is a corresponding node, it might be even larger actually. It might have 3 keys ,this might also be red. [Hindi]. This however is a tricky case, this corresponds to a single key in a 2-4 tree node. So in the 2-4 tree node, if I remove this key then the node becomes empty. That's a problem and here this would correspond to change in the black height.

Hence we can assume that the door deleted is a black leaf that is this is the only case which is really interesting for us and we removing this reduces the black depth of an external node by one and so in general we are going to assume the following. We are going to look at the following step. There is some sub tree whose black height is reduced by one and we want to re organize the tree to take care of that.

Why I am saying sub tree (Refer Slide Time: 31:25) where this is a sub tree. What is going to happen is that as we do reorganizing this tree could become larger and larger. This is a black height [Hindi] and we will see that. So it's going to have a bunch of cases as procedure.

(Refer Slide Time: 34:44)



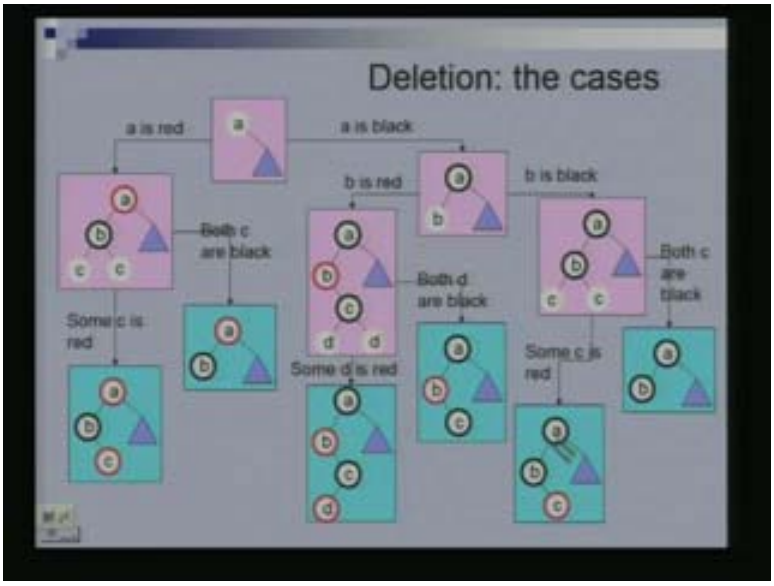
So let me assume that this is a tree [Hindi] black height because of the deletion once [Hindi]. So what I am going to do, I am going to look at the parent. So this is the tree [Hindi] root has sub tree. I am going to look at its parent. The parent is let's say [Hindi]. So first I will check whether a is a black or a is red depending upon this I will have two different cases happening. If a is red so this is a picture. What about the other child of a? It has to be black lets says its b. let's look at the two children of b. Let's call them c.

You do not know whether they are black or red. a is just name for this node nothing else. This is the tree whose black height is reduced by one, the sub tree. I am just looking at the root of the sub tree and its parent (Refer Slide Time: 33:44). So a is the parent. Let me just clarify this point. So what you are saying is we are interested in when we are deleting this black node which has two external nodes and this has some parents somewhere.

When I delete this entire thing I replace it will one external node here. So essentially this sub tree, this node and its two external node children [Hindi]. It is getting replaced by this, black height zero [Hindi]. The problem why we have to formulate this way is that this is going to in some cases what is going to happen is this tree is going to expand. So just take this for now and we will see [Hindi] if you are confused then just think of this has the single node with two children like this picture. You can think of this blue triangle as that, its black height is reduced by one. So this is one possibility. If this is red and this child is black and we do not know what its two children colored are. The other is that if this is black, a is black then b could be red or black.

So depending upon what the color of b is we will have different cases here, depending upon what the color of c is we will have different cases. So these are two cases either both of these are black. That could be one case.

(Refer Slide Time: 37:30)

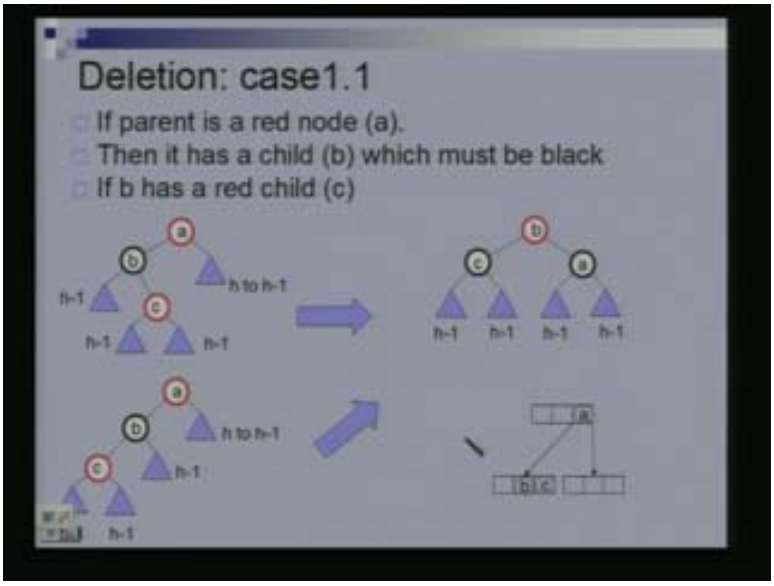


So I have not shown them black, I am just showing you what the various cases are then we are going to handle these cases separately one after the other. So one case when both of these are black, the other case is when one of these is red. This is let's say red and these is one case for us. This is just a part of the tree.

There would be a left sub tree here, there would be some sub tree is hanging here and so on and of course this will have something, this will have a parent and more happening here (Refer Slide Time: 36:28). This is the case when both are black so I have not shown them here but this will be another case. Let's come here. Now depending upon what b is we will have two case red or black. So b is red then this is the red node then it will have a right child which is black, it's right child cannot be a red and this will be black and now I am going to look at two children and depending upon what the two children are I am going to have two more cases. The other option is when b is black in which case we will have two children and depending upon what these two children are once again we will have two cases. So here there is one case when one of this two is red.

So that is the case like this and the other is when both of them are black which means I will just look at the scan structure. Similarly here, there would be one case when both of them are black. When one of them is red then I will have this and when both are black I will have this. So in this manner, this is just a kind of a starting picture to show you that there will be six cases that we will be looking at and we will see what the tree organization, re organization has to be done. So I am going to refer this as case 1.1, 1.2 this will be 2.1.1. So this is the first case we are going to look at. So this is what the picture would be. So a b c and basically a b c and this is the tree sub tree whose height is gone from h to h - 1 let see the black height is gone from h to h – 1.

(Refer Slide Time: 39:53)



Now what will be the black height? So all these heights are now black heights. What is the black height of this sub tree? I have written h – 1. Why is it h – 1? Earlier this had a black height of h, what should be the black height of this? h - 1 because there is one black node here already. So this will have the black height of h – 1.

Similarly this will have black height of h -1, this will have the black height of h -1 and if a node is red then both its children have to be black. Red node cannot have a red child; it's a double red problem. The other possibilities c is the left child of b so it's a symmetric thing. Now how are we going to reorganize this tree? we are going to reorganize this tree in this manner. So it's basically a rotation, b goes to the top, c goes here, a comes there and these have not labeled these things but you can understand 1, 2, 3, 4; 1, 2, 3, 4 in this order will set there.

We will see in this kind of reorganization when you are taking of AVL trees. Now what does this correspond in the case of a 2-4 tree? Let's look at it. So what this corresponds to? This picture is basically that this node was empty that's why the height went down. In the parent node I have an a and it is red which means the parent node in the 2-4 tree has at least two keys. [Hindi] This will be always black [Hindi] why is this rotation? Student: c is greater than b. You are right perhaps, so you are saying a is here [Hindi], so it will become a c and b. So ignore this one, it's for this one, this picture (Refer Slide Time: 41:10).

Now let's see what this corresponds to in the case of 2-4 tree because that's where all of this motivation is coming from. So this is a red node which means its parent is a black. So when I had created the 2-4 tree node I would have at least two keys in the 2-4 need tree node corresponding to this guy, at least two, may be three but at least two. So that is this one, so it could be either two or three I put the a here. Now in the 2-4 tree node corresponding to this guy, actually I took this one, so I would have b and I would have c

(Refer Slide Time: 41:47). so I would have b and c may be there is another key if this guy was red. So this is what the picture is and then what do we do in the case of a 2-4 tree? I have a problem because I don't have any key in this node, what do I do? I borrow from my sibling. My sibling can lend me because it has at least two keys.

So it is going to lend me one. So one of the key is going to go up from here to here and one is going to come down from here to here. So that's exactly what is happening. So [Hindi] one goes up, let's say the b goes up, c remains here. When I am looking at b c, so this corresponds to this. So let me correct this in a second (Refer Slide Time: 43:00). So what we are saying? Suppose we were to keep this one in picture. So lets remove this one now and then this should be c and this should be b.

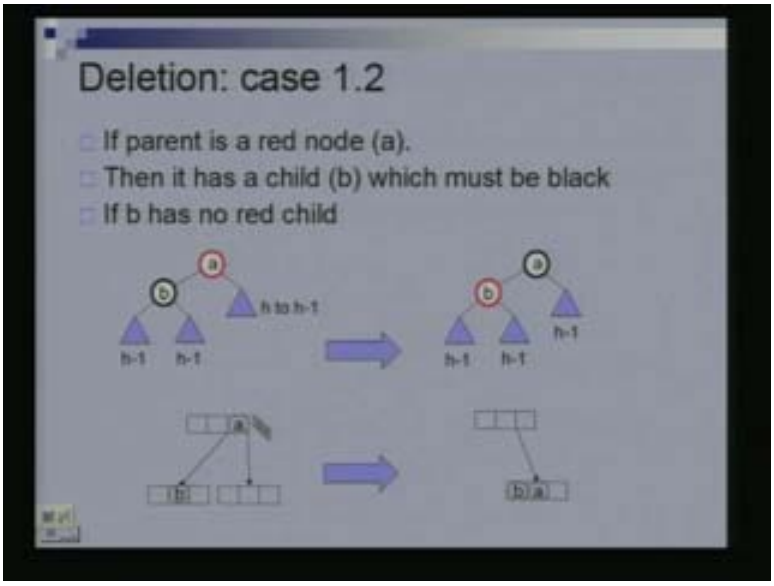
Now it's okay. So let's keep this in mind. Now what's going to happen, we have c and we have b here so which is the one which is going to go up? c and a is going to come down. So in the new node that I get here, I would have only one key a. In node that I have here c is not going to be there and in this node a will get replaced by the c. In this node a is the only key which means a is black. In this node b is one of the key and if there were one other key here either b is only key in this node or if there was one other key here then it is already sitting here, we don't have to worry about that. So b becomes the black node and this c why should c be a red node because there are more keys here. So c can continue to be a red node. so this is just for motivation but if you have look at this. This is good enough.

We have taken care of the height problem, this is my new re coloring, this was red, this is also red so they cannot be a double red problem. Why? Because there is a double red problem if the parent is red but the parent is not red. If the parent is red then there is a double red but if the parent is red then the parent of this was also red which means there was already a double red problem. So there is no double red problem. Let check the height business. So black height is h -1 and now what is the new black height of this sub tree?

So for all these external node it will be h, for all of these also it will be h, for all these will be h, for this it will be h, the black height is h (Refer Slide Time: 45:15). What was the black height of this guy? h - 1 + 1, h - 1 + 1, h – 1+ 1 and this was h to begin with [Hindi]. So the problem is taken care of. You have not introduced double red and the black height has been restored. What case was that? So that was very first case. Let me just show you so it was this one (Refer Slide Time: 45:58), so we have to go through this one (Refer Slide Time: 46:02). So we quickly now start going through them (Refer Slide Time: 39:53).

This is when parent is a red node, so this is what is written. So parent by parent I mean this is a sub tree whose height is decreased, I am looking at the parent of this root it's a red node. It has a child which is black then black child has the red child. This is what we do. So the other thing is when it does not have any red child, b does not have any red child that would be in a second case. So let's look at that.

(Refer Slide Time: 47:55)

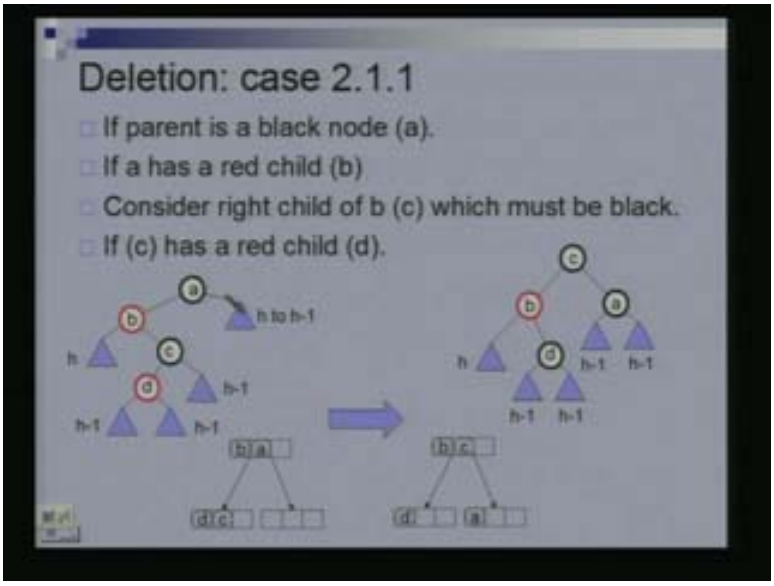


So b has no red child. In this case the picture is a b and both of this sub tree have as the root black children. Now this went from the h to h - 1. So this has to have a black height of h - 1 and this is also have a black height of h -1. Why because originally black height of entire tree was h. So this must be h - 1, this must be h - 1. Now what are we going to do? Just the re coloring will solve the problem because I make this red and make this black. What is the black height of the resulting tree now? H, which is what the original height was. Why this was not introduced the double red? I made this red, no this could be a red. That's why we have taken care of the other case first.

If any one of these where red [Hindi] and what does this corresponds to for our 2-4 tree picture? This is the node [Hindi], this is the only key in this node sitting all alone and then what do we do? If this goes up then this guy becomes empty. So what do we do then? Merge, these two combine in to one single node and one key comes down from the above. So what will the key's in the new node be? b from here, a from here which comes down from the above. So this new node is going to have b a in it which is what is being done here, this new node has b a in it. This will be one node, I take a black and look at all its red children. This will be one node with be a in it.

So this is one to one corresponding between what we did in the case of 2-4 tree and what we are doing here and that helps to clear things. So we are looked at both the cases when this guy, when the parent was the red if then the parent was the red then we know this is a black. If this had any red children we have taken care of it in previous case, if this has no red children then we taken care of it now. So now we go to the red next set of cases when this parent is a black. So parent is a black node.

(Refer Slide Time: 52:54)



If parent is a black node then this could be a red or could be a black? So first considering the case that it has the red child b. If this is red then this has to be black and if this is black then this may be red may not be red. Now again some two condition I am making or some two assumption making of; this is red first and this is red. In the next case I am going to consider this was not red, this was black and this was black. This node had no red children and you see why this is required, you already seen one reason why to consider whether node has red child or not.

So this is a picture. This height went down from h to h -1. So what is the black height of this tree? This entire black height that means was originally h +1. [Hindi] first let's make sure that this is correct and then we will see why we came up with this and why we came with this is justified by our 2-4 tree. So let's just check it is correct. So first you can see here that b is less than a, b is on the left c.

C is more than b less than a so c lies between b and a and d is more than b but less than c, so d is more then b and less than c. So search property is okay and 1, 2, 3, 4, 5; 1, 2, 3, 4, 5, I just organize as before. They just go in a same manner. Now this one is a this one, it has now black height of h -1. This guy is this, its black height of h - 1 and this one is a very first one, its black height of h. So now if you look at the black depth of the external node so these guys will have black depth of h + 1. this will have h minus one plus one plus one, h +1 h -1 + 1 + 1 h -1 +1+1 (Refer Slide Time: 52:15).

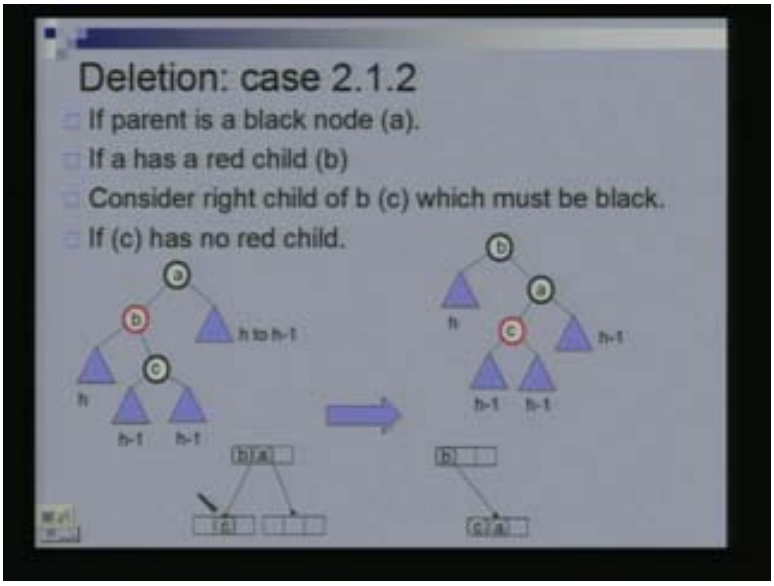
So the black depth of all the external nodes are the same so that thing is taken care of and far as a double red problem is concerned the only thing could be that this is red [Hindi]. It was only a red black tree and it had only made changes in this sub tree. So this is a valid thing and in takes care of the problem. Now where is this coming from? What is the motivation? Again this was an empty node, this corresponds to an empty node, this is a

parent node, this has a and this has red child so it means that two keys a and b and this has one child here which has two keys c and d.

So that's what happening here. This corresponds to this node and a b corresponds to parent node and what happens now? I can, because my sibling has enough keys it will lend me one. So one of the key will go up and a would come down. Which is the key which would go up, the largest one. C would go up and a would come down, this is the picture would have b c d a and what is this corresponds to? This corresponds to exactly this b c here corresponding to this node, a this one and d this one. So everything is coming from the 2-4 tree. What would you have done in this case? This is direct one to one correspondence, you just use this thing to decide what to do here.

So what is the case we considered here that this is black. We also considered the case when this parent was the red and then we made the assumption that it has the red child then this has to have the black child and we made the assumption that this is red. So now we will get rid of the last assumption that this is red that means this guy has no red child, c has no red child which means both of these are black (Refer Slide Time: 54:20). So we come to that key.

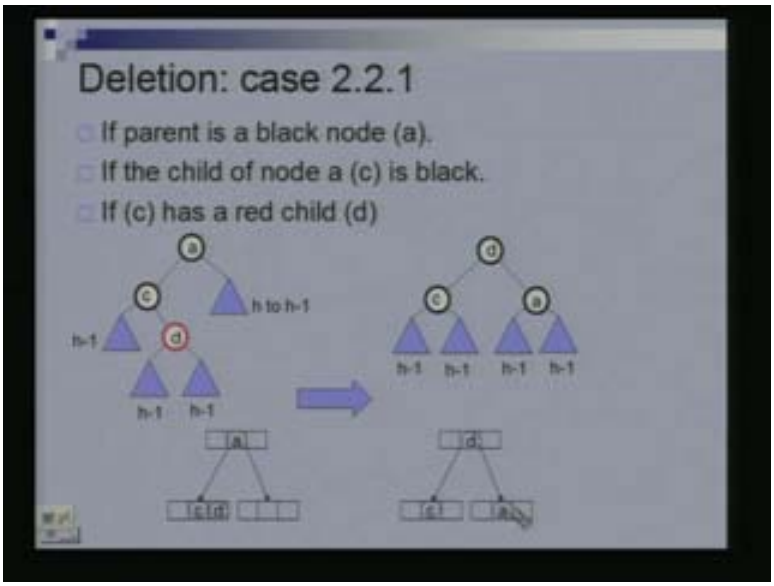
(Refer Slide Time: 56:03)



c has no red child and so this is the picture. So this are two sub tree both of whose two root are black and so this has to have black height of a h -1 because the entire black height was h +1 to begin with same as before [Hindi] and we will see the motivation once again coming from the 2-4 tree. Though once again c is between b and a so b here, a is larger than b, c is more then b but less then a so binary search tree property is okay and the same t_1 t_2 t_3 t_4 [Hindi] t_1 t_2 t_3 t_4 [Hindi] black height check [Hindi] black depth [Hindi]. Double red problem [Hindi], these are both black [Hindi]. Where is this coming from once again? This is the node which is getting empty. The parent has a and b in it and then this has one child which has only c in it because both of its children's are black. So now

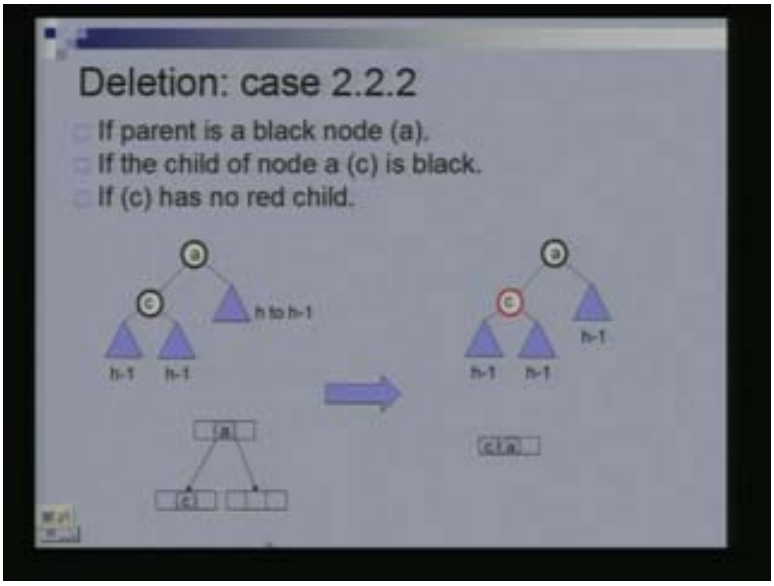
once again we cannot borrow, we have to merge and merge may c [Hindi] which corresponds to this [Hindi]. So c has no red child. Now what is the case left? We started assumption that the b is red. So now we need to work with assumption that b is black that was assumption b is red. So we come to b is black [Hindi]. This is the node we are talking of, this is black.

(Refer Slide Time: 58:07)



So now again the next case that it has a red child and one possibility and the other is it does not have a red child. So if it have a red child lets say this is a red [Hindi] d is between c and a. So d is between c and a, where is it coming from? This is empty, a does not have any red child which means its only key in its node and this has a red child so that means there are two here. So once again it is going to borrow one from here. So d is going to go up, a is going to come down and we will have this picture, so d is the only key in its parent, this may or may not be the only key in its node because this could also be a red but we are not bother about this [Hindi] but this only be the one key in its node because this node was earlier empty [Hindi] c does not have the red child at all. a c c no red child[Hindi].

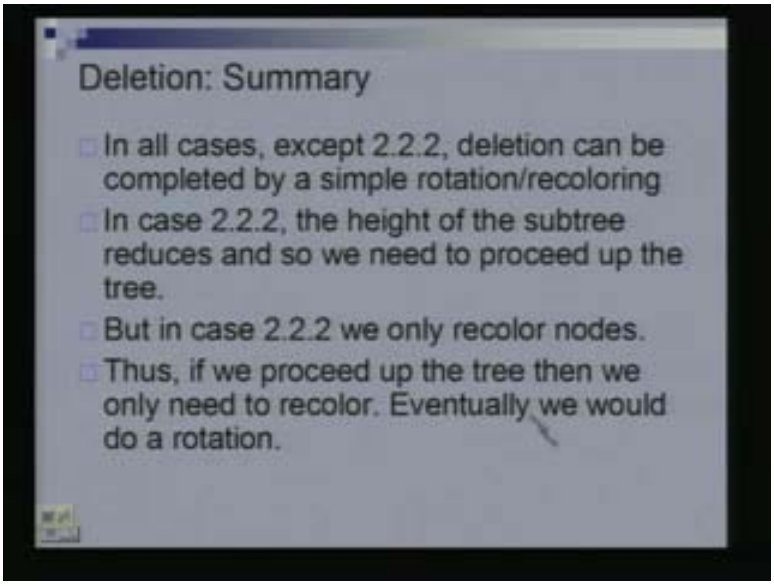
(Refer Slide Time: 1:00:00)



So now what do we have to do? [Hindi] h -1 +1h [Hindi]. I worked with the assumption that this is the sub tree [Hindi] black [Hindi]. What case does this correspond to? This has only one single key, this neighbor also has only one single key. So you kind of combine both of them, they come here and the parents becomes empty. so the parents becomes empty so you have to then repeat the process for 2-4 tree deletion in the parent which means it either has to borrow from one of its siblings or it has to merge with one of its sibling and all of that. So there was the same kind of thing happening in the case of 2-4 trees there.

That the process continued up, the deletion process. The parent became empty now so you have to do something there and may be then again the parent became empty so you have to do something there and so on and all the way up to the root in which case you reduce the height of the tree at the end of the thing. The same thing could be happening here except that here all we are doing is re coloring a node. [Hindi]. Essentially one bit of information [Hindi]. In one of the six cases we have to go up but when we have to go up we don't have to do too much work. We just have to do one recolor. So that's the summary. So in all cases except the last case 2.2.2.

(Refer Slide Time: 1:01:45)



So deletion can be completed by either some reorganizing, by some rotation or by some re coloring [Hindi]. In this particular case the height of the sub tree reduces and so we need to proceed up the tree but in this case we are only re coloring the node [Hindi]. So what is happening is why is this is a fast procedure because you have to do essentially one rotation only [Hindi]. It's a very short, so which is what makes the process really fast except for the last case in all case the height preserving [student] [Hindi].

You have to check that [Hindi], you have to check this last thing if you have to convince but this is the entire process of deletion. Next class we are going to look at the process of insertion, it's much simpler than this. So this was the harder one but keep in mind you don't have to remember it, if you remember the 2-4 tree which was much simpler to understand conceptually. If you remember that you will also be able to remember this process.