



Red black trees

Contents:-

Red black Trees

Height of Red black Trees

Red black Trees to (2,4) trees

(2,4) Trees to Red black Trees

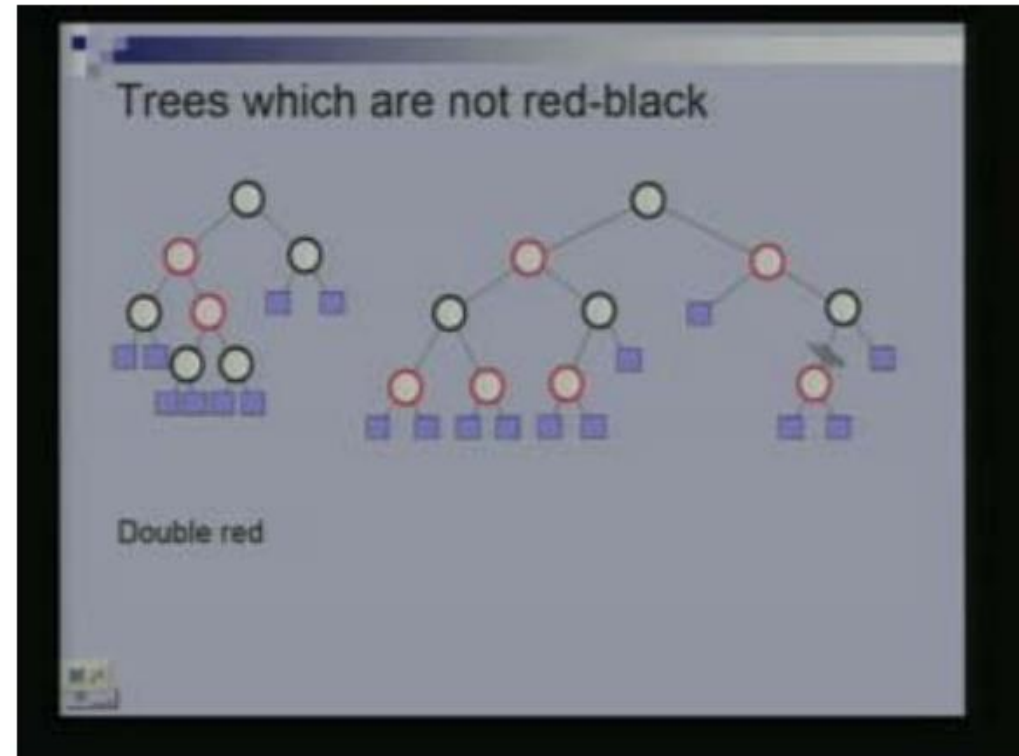
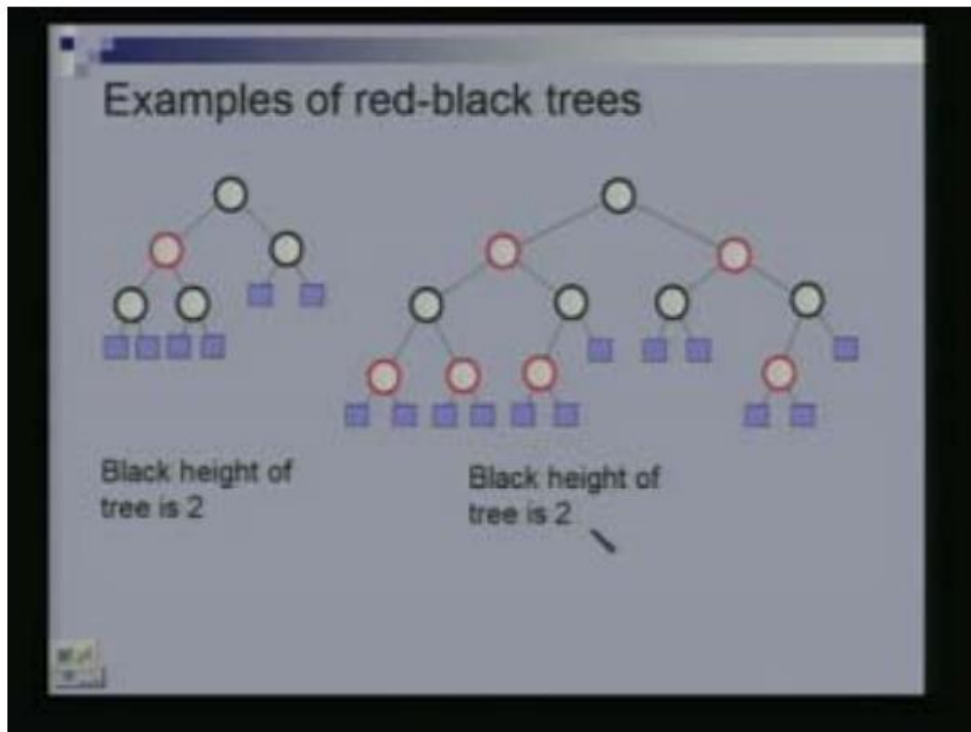
Deletion in red black trees

Insertion in red black trees

Introduction

- Red black tree is a binary search tree.
- 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.
- The black depth of an external node as the number of black, its number of black ancestor's.
- 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.

Red black tree examples:-

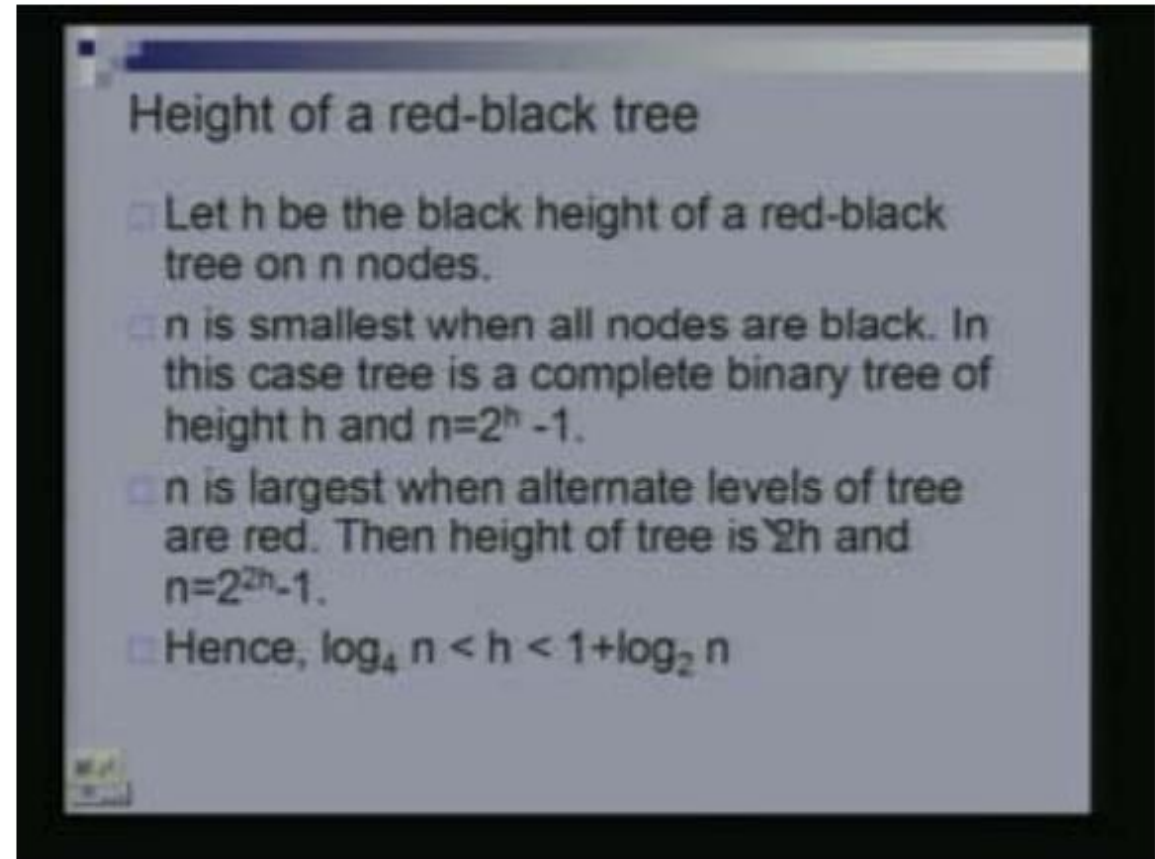


Height of red black tree in worst and best case

- Height of red black tree is

$$\log_4 n < h < 1 + \log_2 n$$

This is similar to that of (2,4) tree.

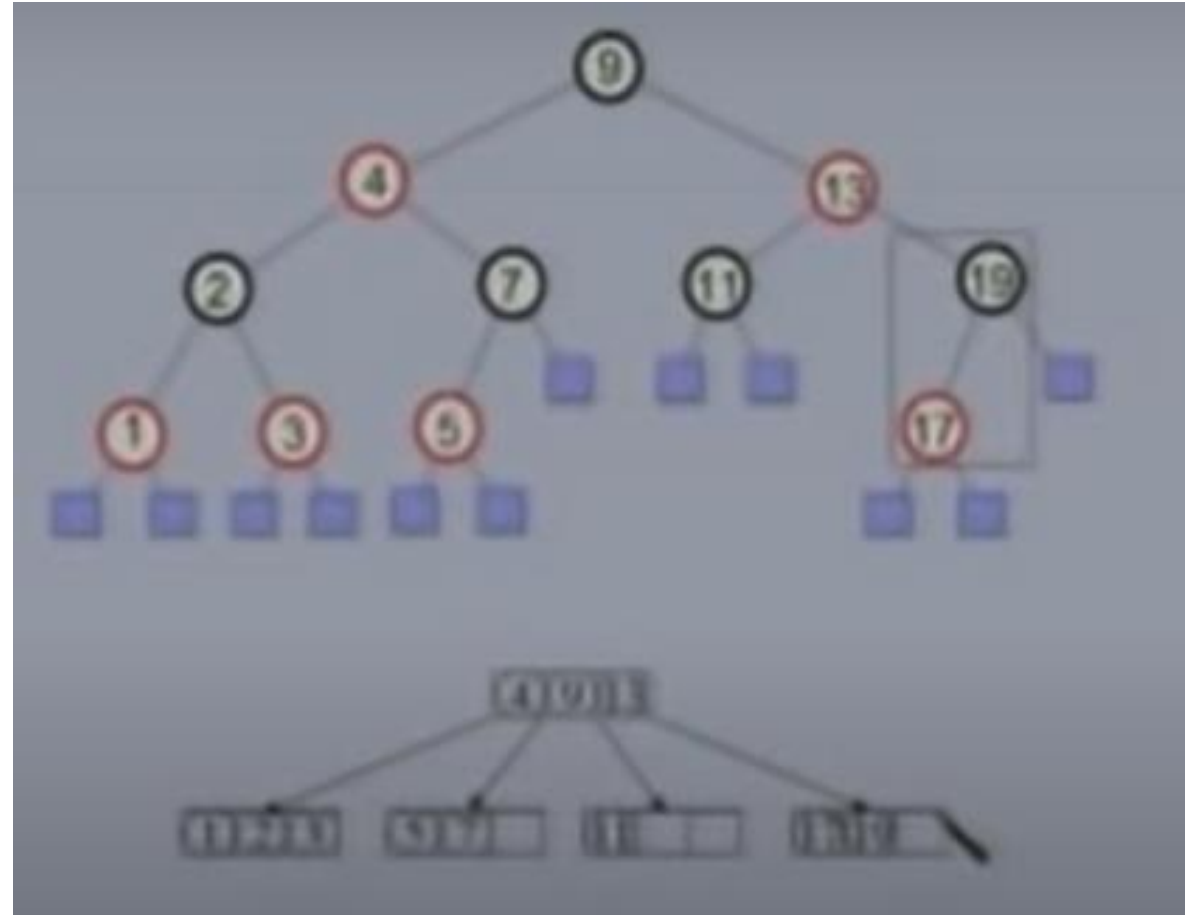


Height of a red-black tree

- Let h be the black height of a red-black tree on n nodes.
- n is smallest when all nodes are black. In this case tree is a complete binary tree of height h and $n = 2^h - 1$.
- n is largest when alternate levels of tree are red. Then height of tree is $2h$ and $n = 2^{2h} - 1$.
- Hence, $\log_4 n < h < 1 + \log_2 n$

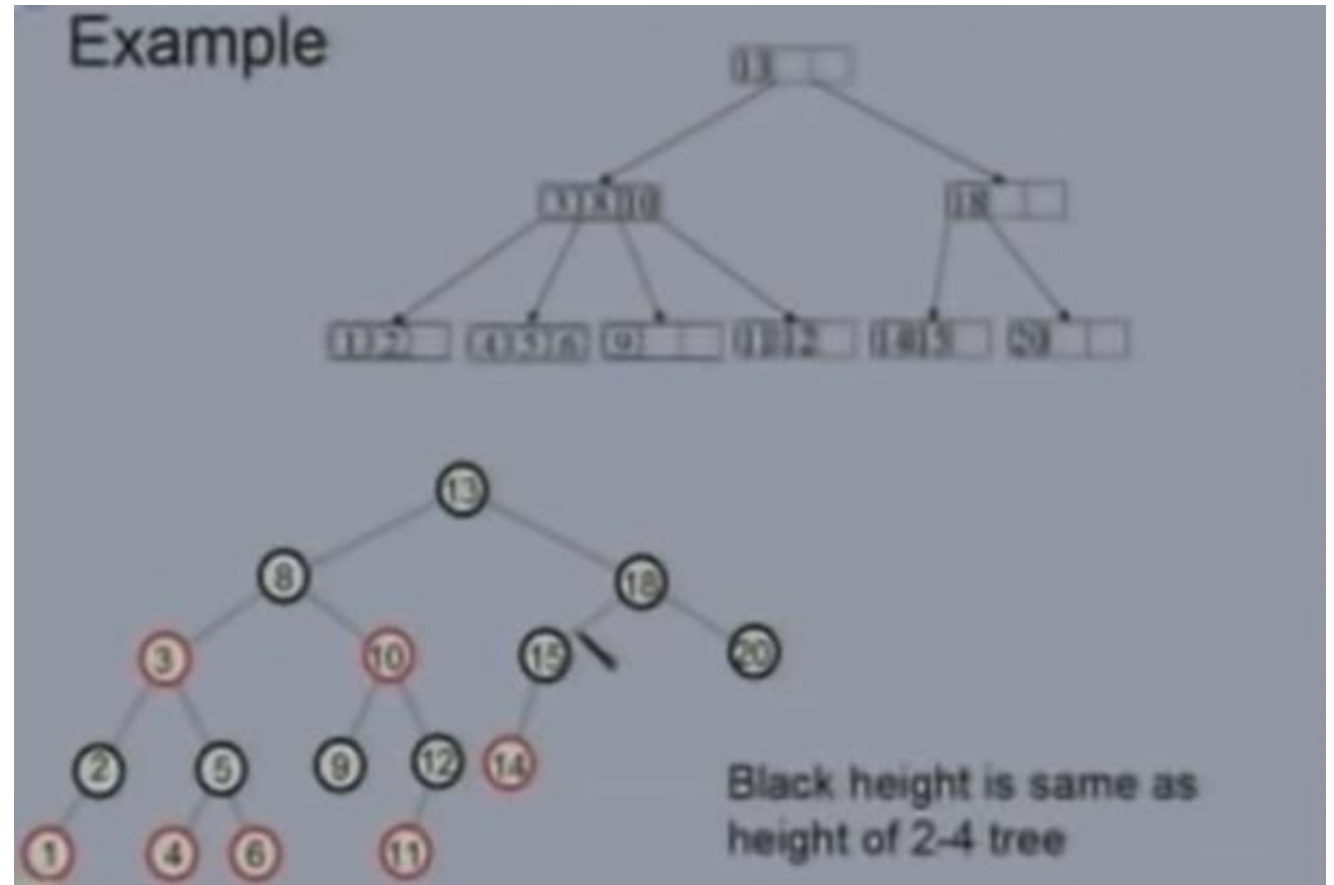
Red black tree to (2,4) trees

- Any red black tree can be converted in a (2,4) tree .
- 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
- black height of all the external node is the same, so in the resulting 2-4 tree all leaves will be the same.



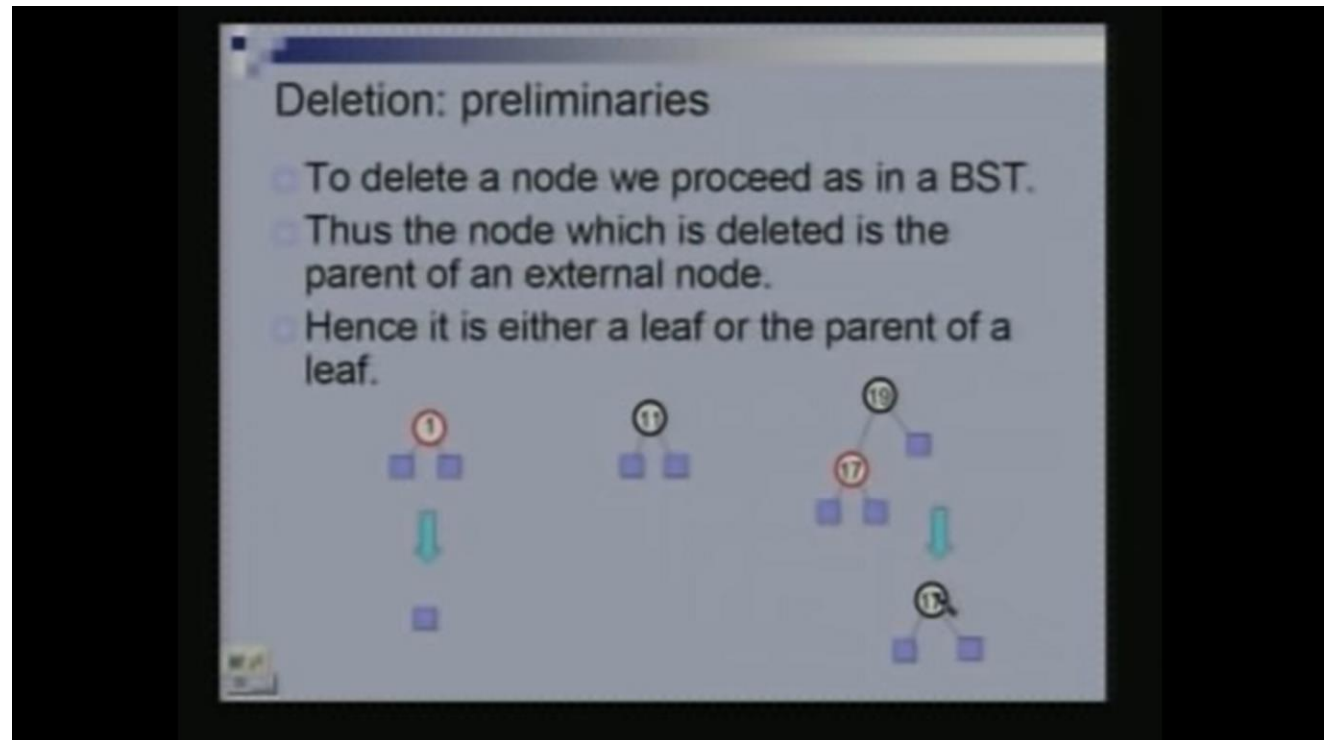
(2,4) tree to red black trees

- Any (2,4) tree can be converted in a red black tree .
- Take a node of the 2-4 tree, replace it with 1 black node and then appropriate number of red nodes
- 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
- 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.



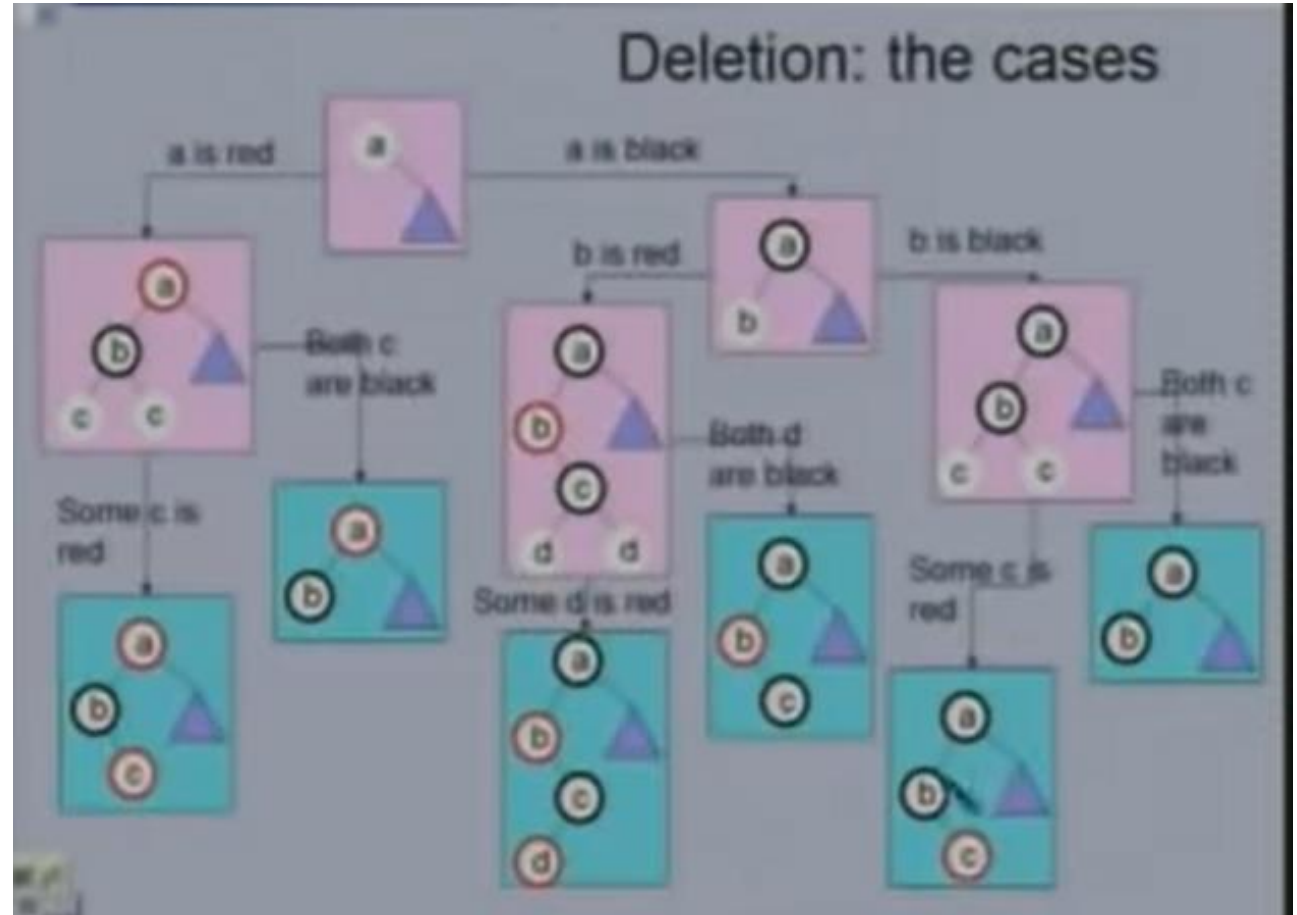
Deletion in red black tree

- There can be these 3 cases for deletion in red black tree.
- For 1st case we can simply remove it as no black height is disturbed.
- For the 3rd case we need to swap the internal node 19 with 17 as the black height will be disturbed and after that we can remove 19.
- The 2nd case is tricky as the child nodes can be red or black both so we will discuss it in depth.



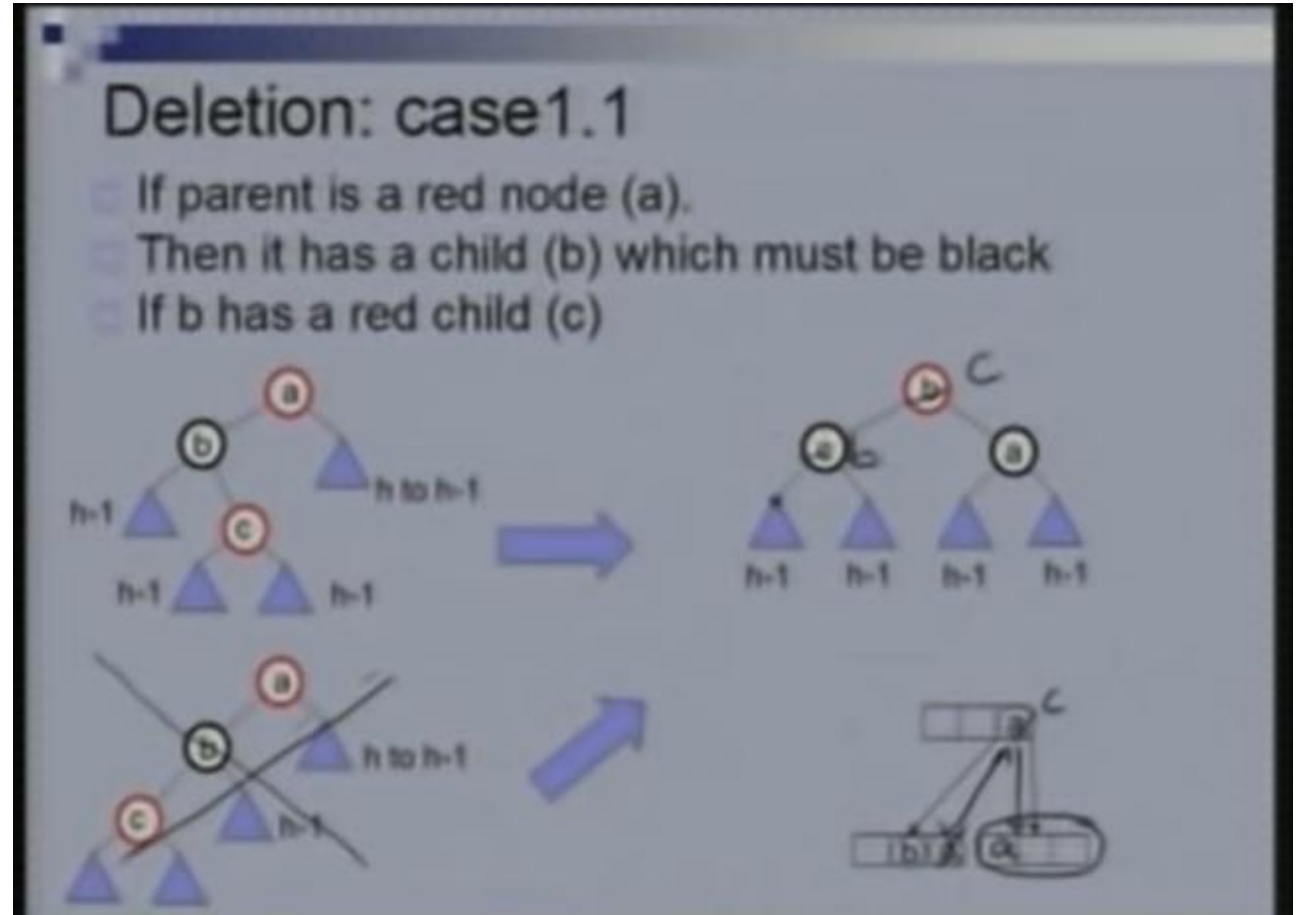
Case 2 in detail:-

- Hence we can assume that the door deleted is a black leaf and removing this reduces the black depth of an external node by one.
- The triangle here represents the subtree in which the black node is deleted .
- So there can be 6 cases possible after this deletion that we need to resolve.



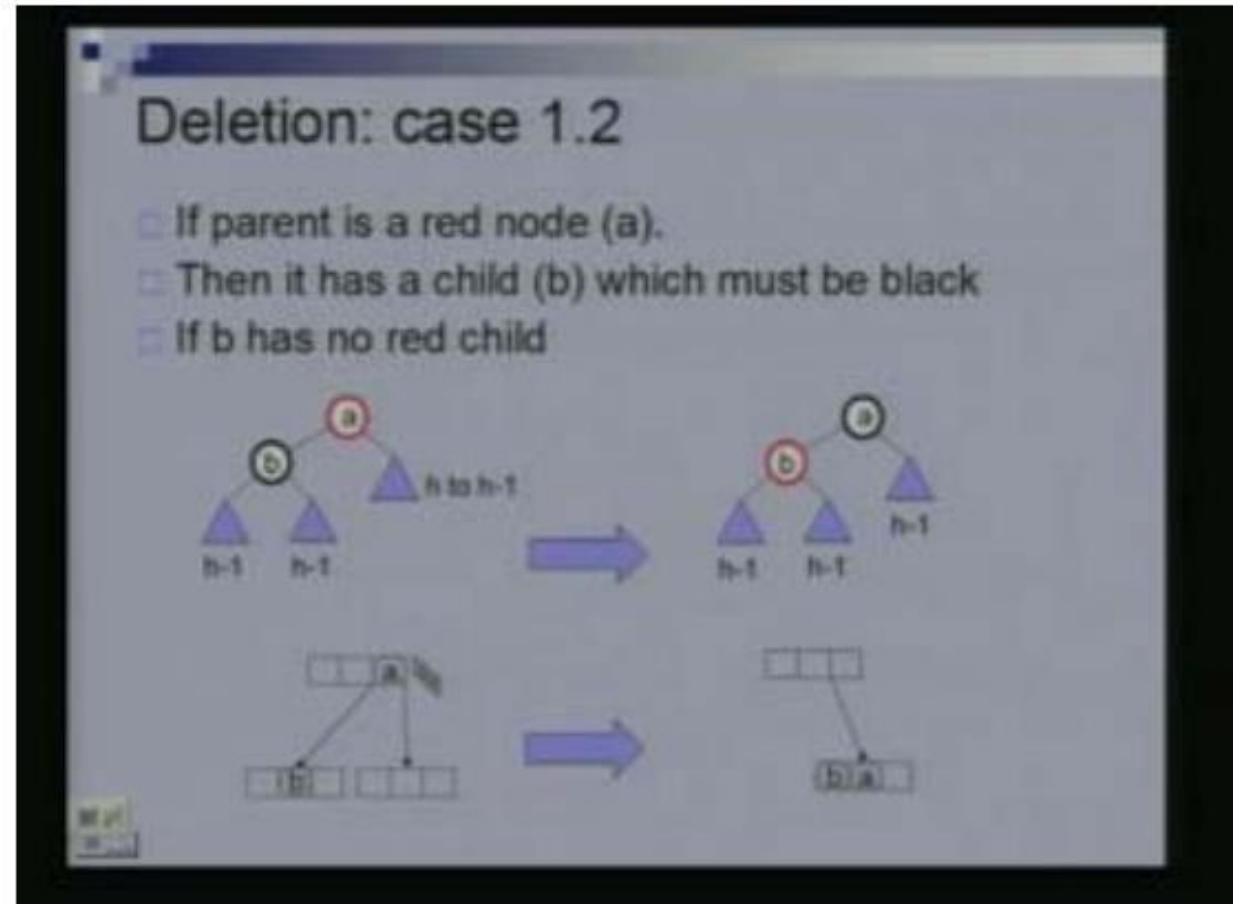
Case 2 (1.1):-

- Hence we can assume that the node a is red then b must be black
- And b has a red child c .
- The subtree below A had height h but now it is $h-1$ as we have deleted .
- And other sub trees have height $h-1$ because b is black.
- So what we will do is like and avl rotation which we also performed in 2,4 tree to resolve the black height.



Case 2 (1.2):-

- Hence we can assume that the node a is red then b must be black
- And b has no red child .
- The subtree below A had height h but now it is $h-1$ as we have deleted .
- And other sub trees have height $h-1$ because b is black.
- So what we will do here is just recolor A as black and B as red
And the issue is resolved.

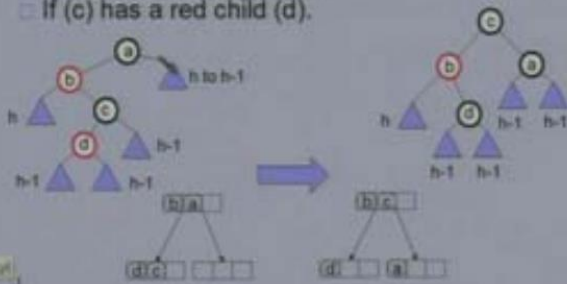


Deletion cases :-

- These are more possible cases that can be there and all of them will be resolved the some kind of rotation to balance the black height of tree.

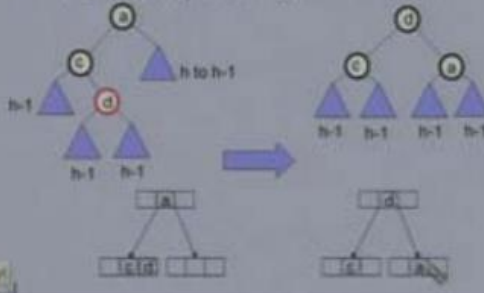
Deletion: case 2.1.1

- If parent is a black node (a).
- If a has a red child (b)
- Consider right child of b (c) which must be black.
- If (c) has a red child (d).



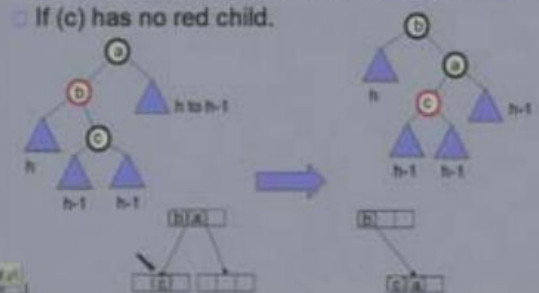
Deletion: case 2.2.1

- If parent is a black node (a).
- If the child of node a (c) is black.
- If (c) has a red child (d)



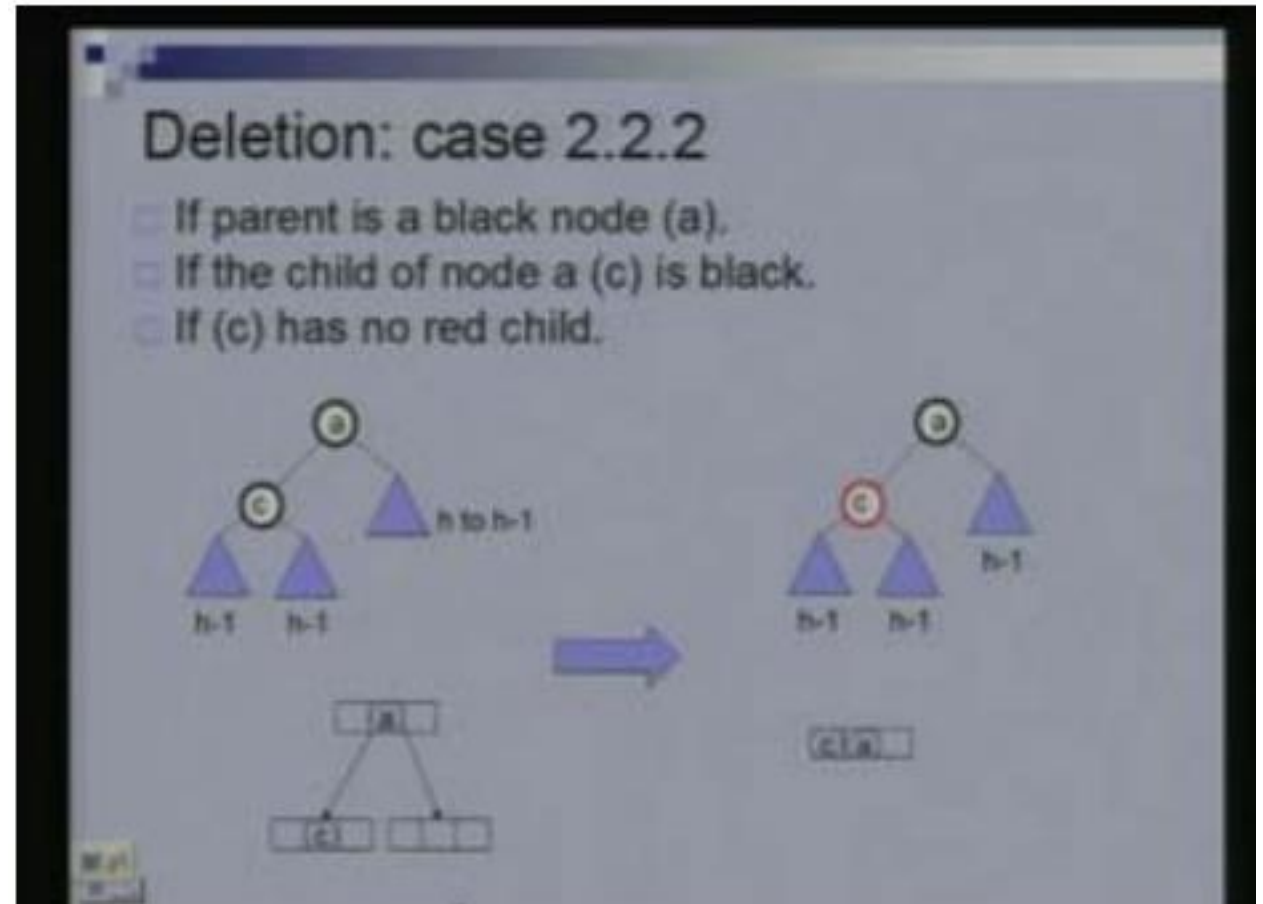
Deletion: case 2.1.2

- If parent is a black node (a).
- If a has a red child (b)
- Consider right child of b (c) which must be black.
- If (c) has no red child.



Case 2 (2.2.2):-

- Hence we can assume that the node a is black and child of a is black and c has no red child.
- The subtree below A had height h but now it is $h-1$ as we have deleted.
- And other sub trees have height $h-1$ because c is black.
- So what we will do here is just recolor C as red. And the issue is resolved. But there might be more to this if this whole tree with a as root node is a subtree itself then the height issue can cascade.



Deletion Summary:-

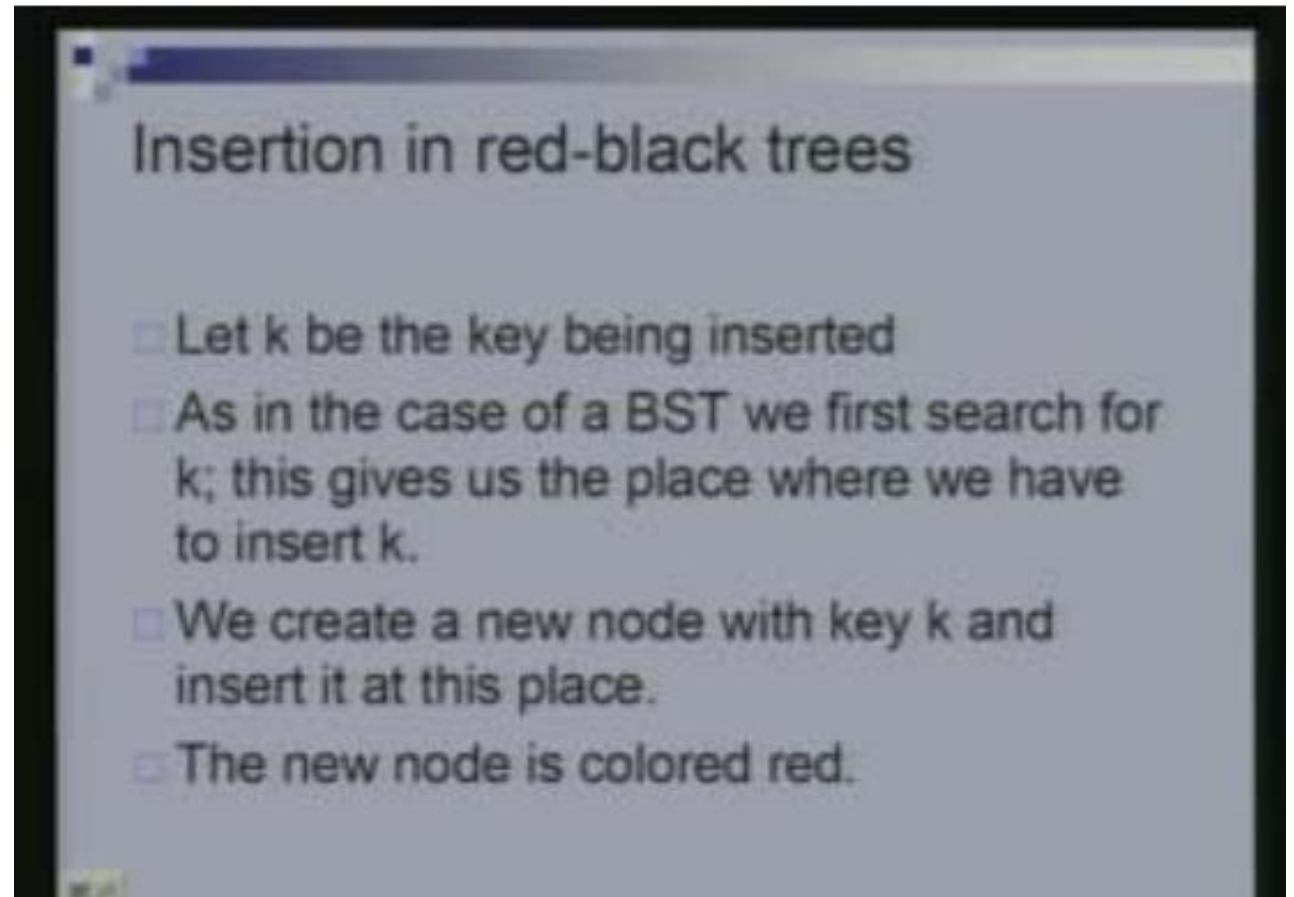
- So we can resolve the black height mismatch issue by either rotation or by recoloring the nodes.
- So this is why this process is very fast because the rotation and recoloring processes are very fast.
- And except for the last case every deletion case was over by either 1 rotation or recoloring

Deletion: Summary

- In all cases, except 2.2.2, deletion can be completed by a simple rotation/recoloring
- In case 2.2.2, the height of the subtree reduces and so we need to proceed up the tree.
- But in case 2.2.2 we only recolor nodes.
- Thus, if we proceed up the tree then we only need to recolor. Eventually we would do a rotation.

Insertion in red black tree

- So same as in BST for inserting we first do searching and then insert it at location.
- New node is always red in red black tree




Insertion cases:-


- There can be 2 cases while inserting parent is black so than no problem just insert.
- But if the parent is red so double red problem is there we will discuss the further possible cases to this case.

Insertion(2)

- Since inserted node is colored red, the black height of the tree remains unchanged.
- However, if the parent of inserted node is also red then we have a double red problem.



No problem



Double red problem

Case 1:-

- In this case what we do is we perform the rotation where a goes up and we color it black and the b node goes left to it and we color it red because its child is black so we can easily color it red.
- In this way the double red issue is resolved.

Insertion: case 1

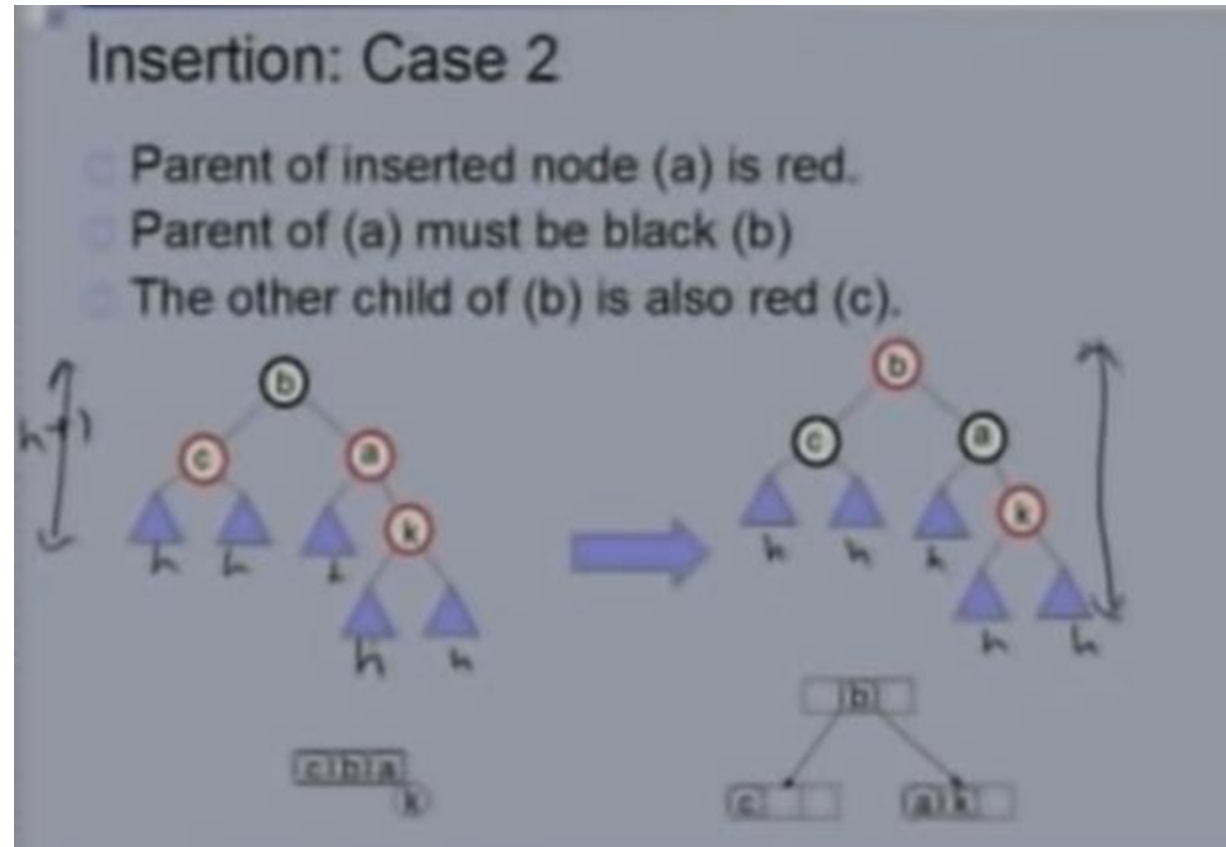
- Parent of inserted node (a) is red.
- Parent of (a) must be black (b)
- The other child of (b) is black (c).



The 2-4 tree node contains (b,a,k) and is malformed.
The rotation corrects the defect.

Case 2:-

- In this case what we do is we just recolor the Node A and C as Black from red. So this resolves the height issue.
- This case is similar to the node split in (2,4) trees. It is mentioned in the diagram .
- But this case can be complicated if the parent of b was red.
- So we Double the double red problem.



Case 2 Continue:-

- So if the parent of b is red then red problem moves up a level.
- And there can be a case we reach the root and color it red then we have to recolor it again to red.
- And in the (2,4) this reponds to splitting of root node.

Insertion: case 2 (contd)

- The parent of b could also be red. In that case, the double red problem moves up a level.
- We repeat this process at the next level.
- Eventually, we might color the root red.
- In this case we recolor the root black. This increases the black depth of every external node by 1.
- In the 2-4 tree this corresponds to splitting the root.

Summary:-

- In Both deletion and insertion we need to make at most 1 rotation
- We might have to move up the tree but in doing so we only recolor nodes.
- Time taken is $O(\log n)$ for all the operations like Insertion, deletion successor ,predecessor etc.
- Most of these operation don't change the tree only insertion and deletion does and we see that they can also be done in $O(\log n)$
- So red black tree is very fast in terms of operations.



Thank you!