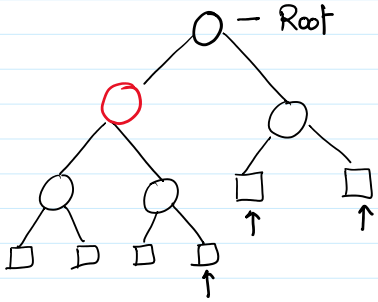
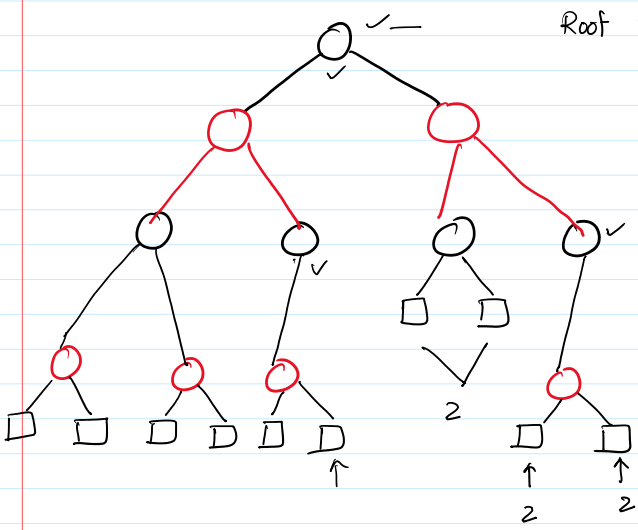
10) Lecture 23-25 Red Black Tree

23 September 2024 22:03

- A red black tree is a BST in which each node is either colored red or black
- Root is always colored black
- Red node can have only black children
- If a node of BST doesn't have left or right child then we add an external node
- External nodes are not colored
- The black depth of an external node is the number of black ancestors it has
- In a red black tree every external node has same black depth

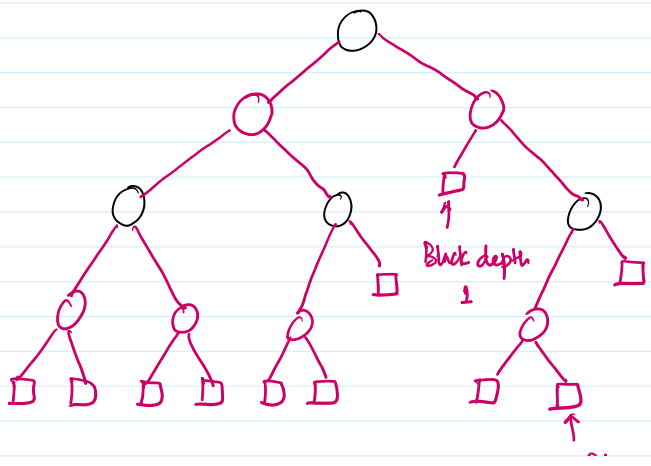


Black depth of all external nodes is 2.

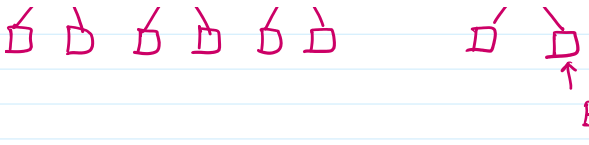


Red Black tree

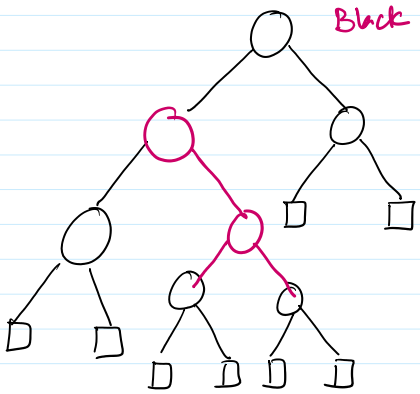
R-R X



Not Red Black tree



Black depth 2



Not Red Black tree

Red Red problem

Height of Red Black tree

Let h be the black height of Red black tree of n nodes.

n is smallest when all nodes are black $\Rightarrow$ $n > 2^h - 1$

n is largest when alternate level are black & red. $\Rightarrow$ Overall height of tree = $2h$

$$n < 2^{2h} - 1$$

$$2^h - 1 \leq n \leq 2^{2h} - 1$$

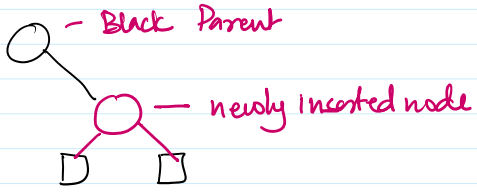
$$O(\log_4 n) \leq h \leq O(\log_2 n)$$

Insertion algorithm for red-black tree

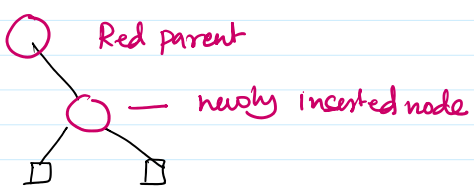
Input: Red Black tree, and node with key.

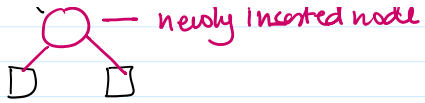
- Let K be the key being inserted
- We create a new node with key value K, and insert it like BST.
- The new node is colored red.

Case (i)

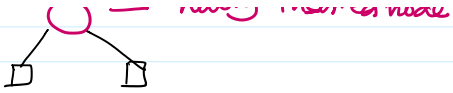


Case (ii)





✓

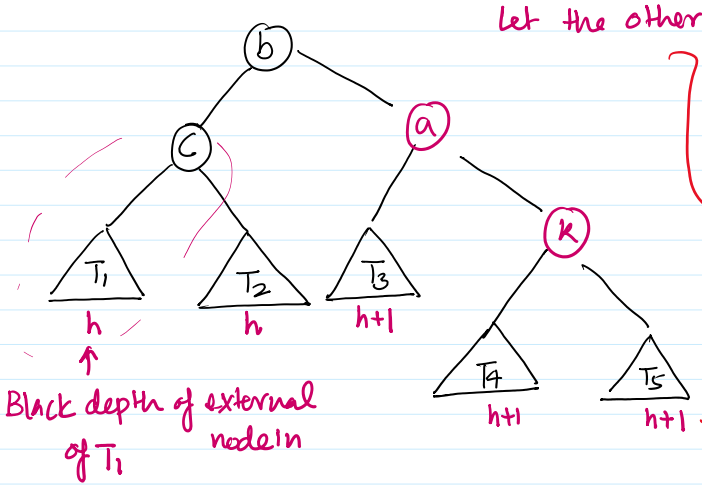


Double red problem

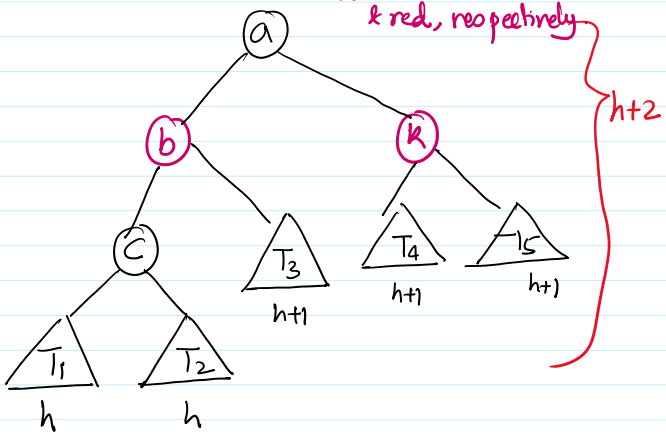
Case 2.1 Parent of inserted node K is red. The parent of ⓐ must be black.
o.w. double red problem

ⓐ

Let the other child of ⓑ, ⓒ is black or it is an external node

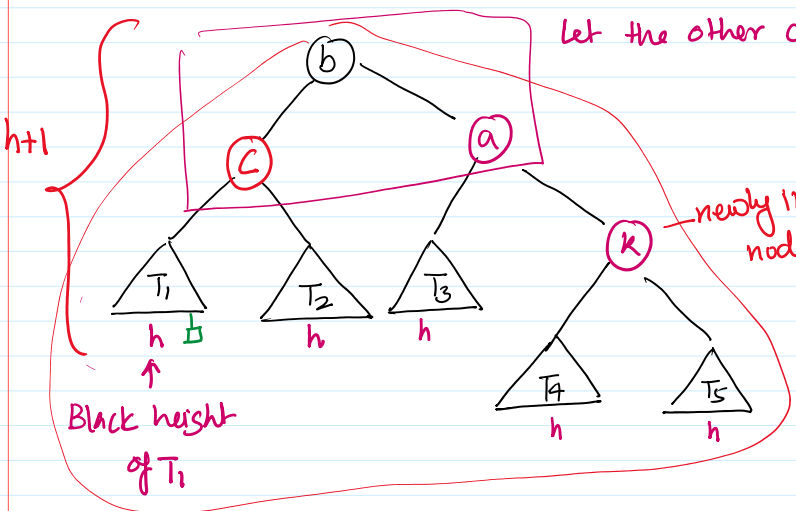


flip color of ⓐ & ⓑ to black & red, respectively



Case 2.2 Parent of inserted node K is red. The parent of ⓐ must be black.
o.w. double red problem

Let the other child of ⓑ ⓒ is red.

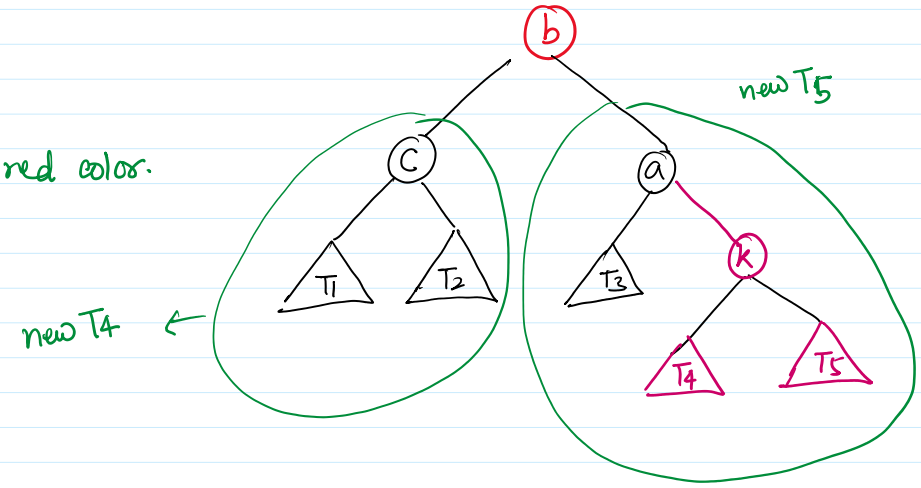


ⓑ

of T_1

if parent of (b) is of red color.

(b) → new (k)



- The parent of b can also be red.. In this case double red problem moves up a level.
- We repeat this process & eventually might have to color root node as red.
- In this case, we recolor root node as black.. This may increase black depth of all external node by 1

Running time: Worst case running time $O(\log n)$

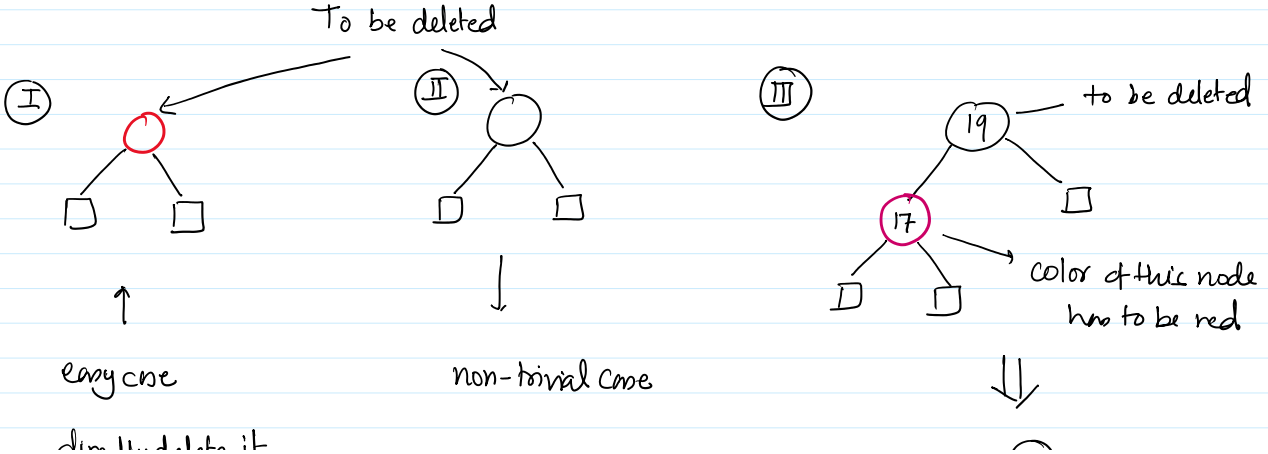
Worst case input:= keep on falling in case 2.2.

Deletion in Red-Black tree

If the node to be deleted is leaf, then directly delete it.

Otherwise we first swapped with its successor and then successor node is deleted.

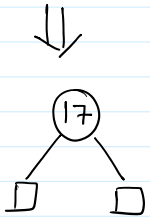
⇒ The node that is deleted is either leaf node or parent to a leaf node



easy case

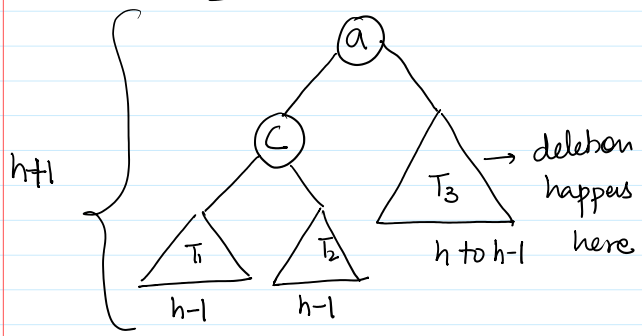
directly delete it

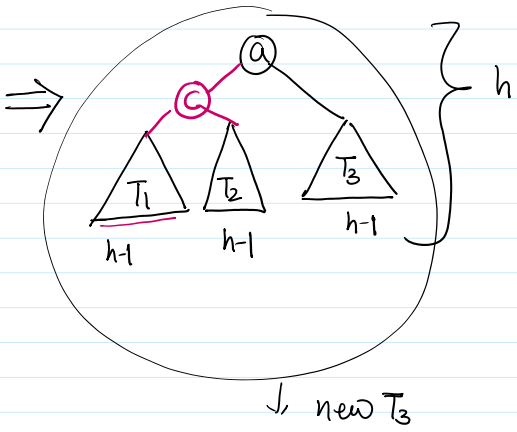
non-trivial case



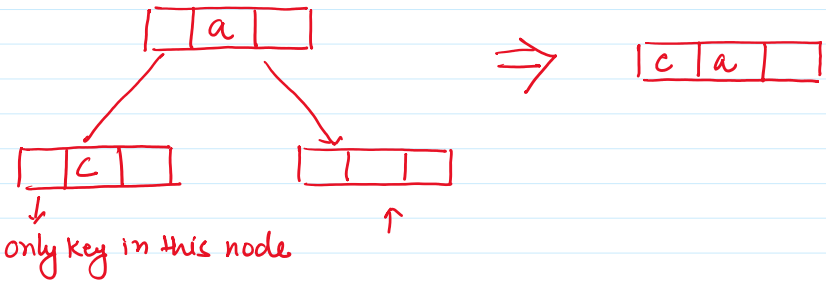
Deletion Cases: Parent is black node (a). Child of (a), (c), is black. (c) has no red child.

1.1

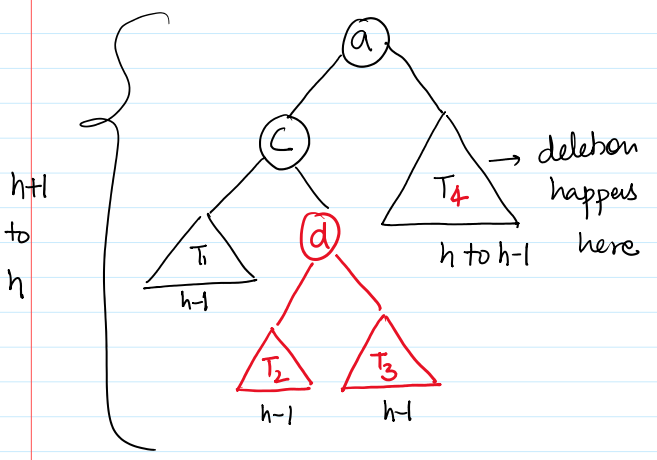


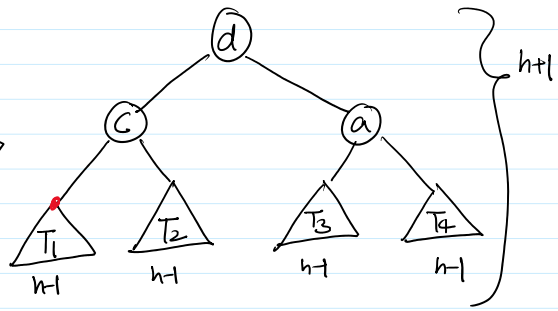


Black depth of all external nodes in T_1 is $h+1$ w.r.t. T_1 only

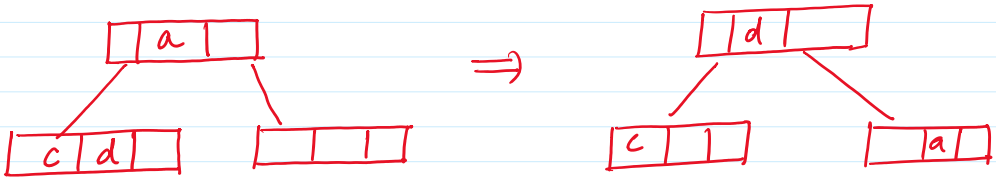


Case : 1.2 Parent is a black node (a). Child of (a) (c) is black. (c) has one (or two) red child (d).

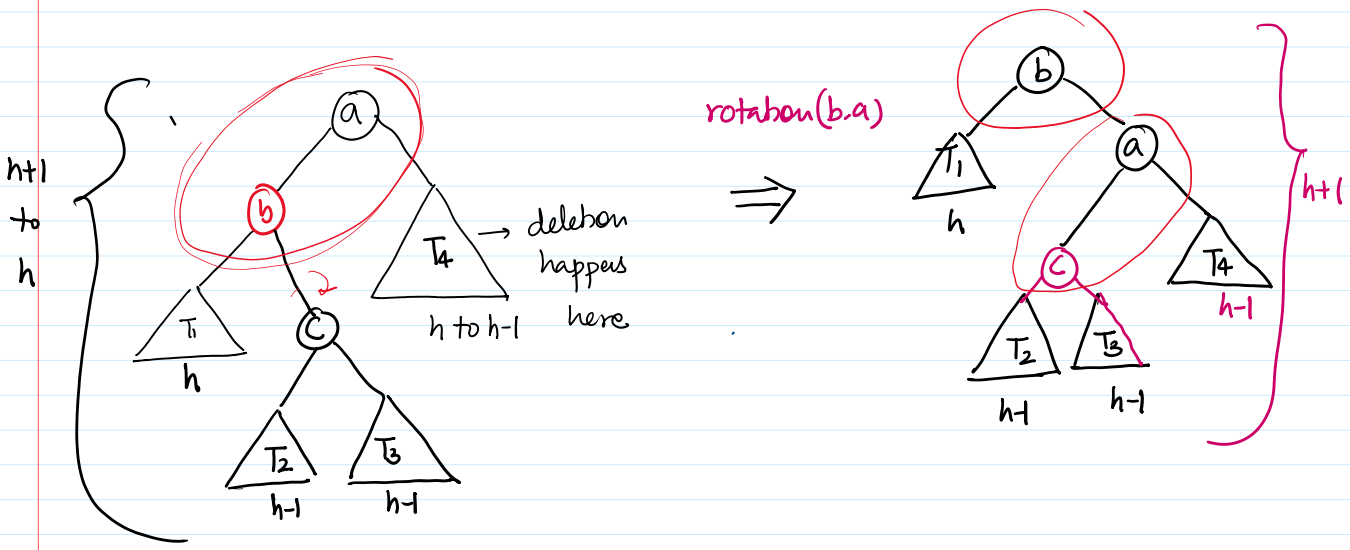


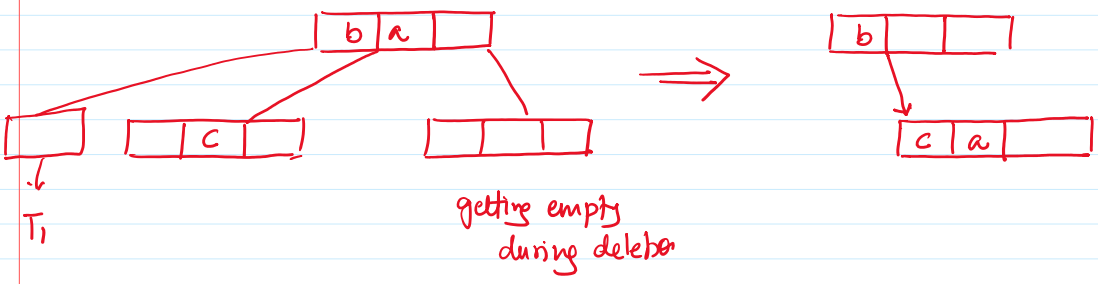


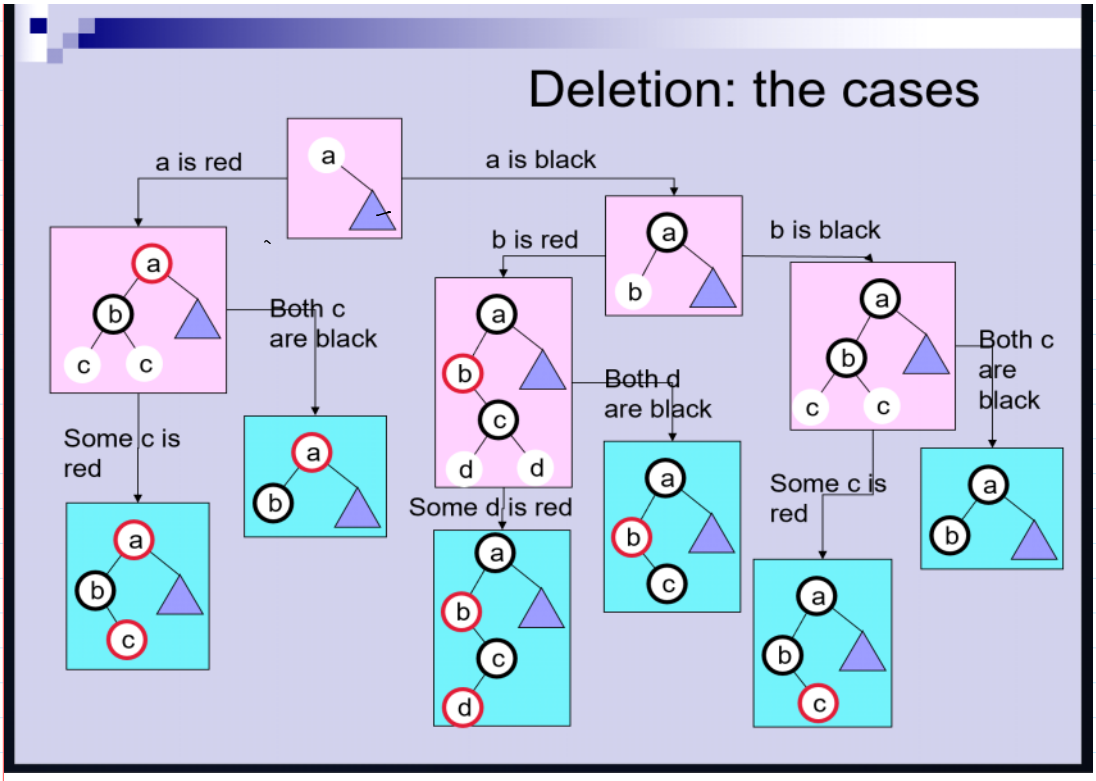
No need to proceed further!!



Case: 1.3 The parent (a) is black, (a) has red child (b). Consider right-child of (b) i.e. (c) that must be black. (c) doesn't have red child.

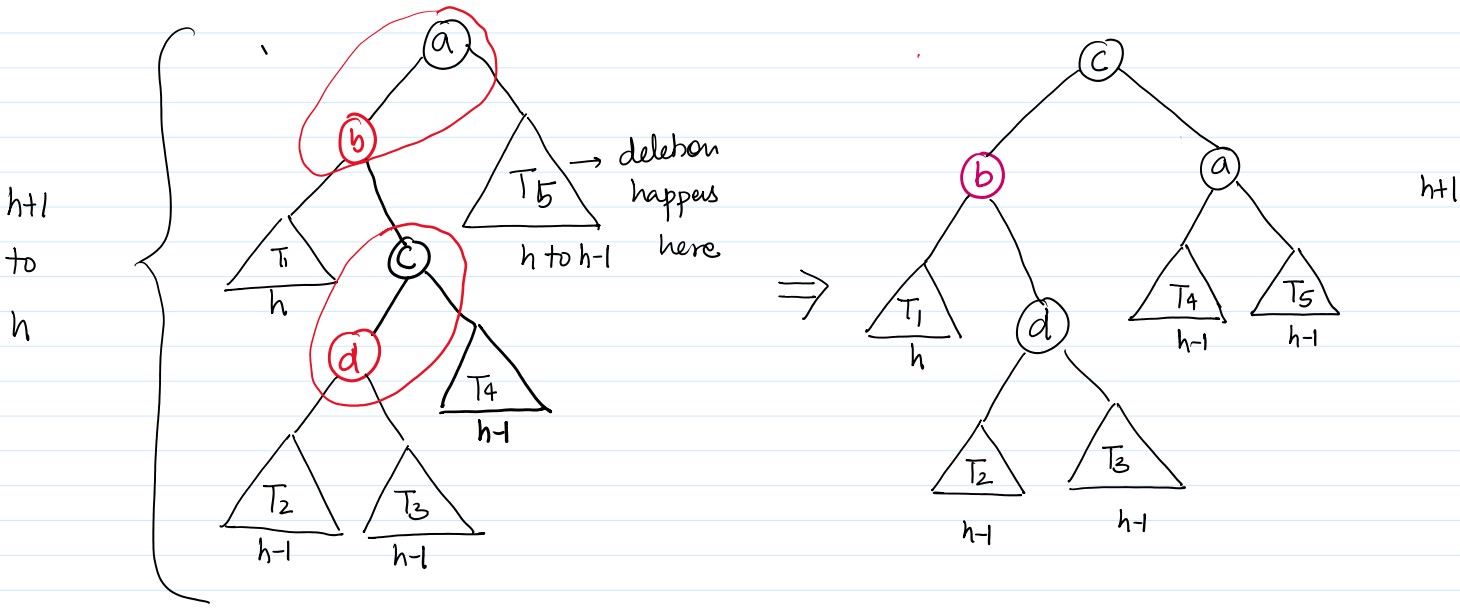


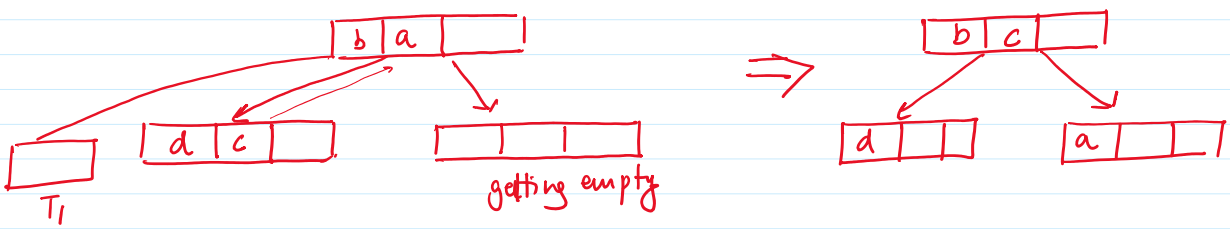




Case: 1.4

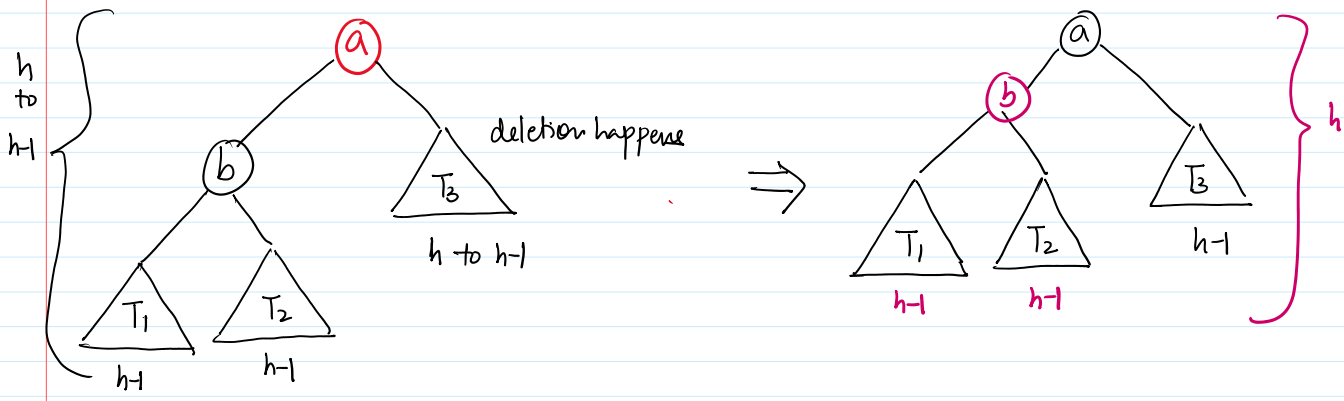
The parent (a) is black. (a) has red child (b). Consider right-child of (b) i.e. (c) that must be black. (c) has a red child (d)



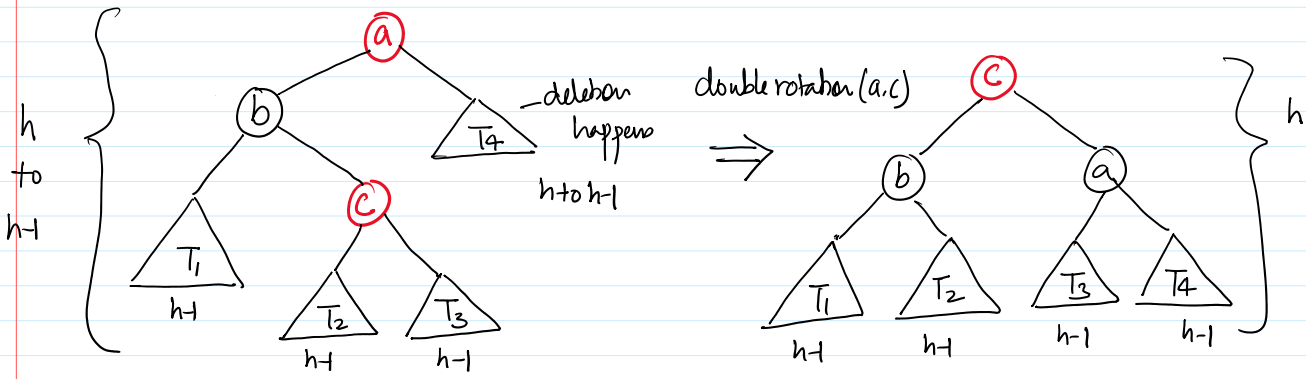


Case: 2.1 If Parent (b) is red node Then (a) has child that must be black.

Case: 2.1 If parent (a) is red node. Then (a) has child (b) that must be black (b) has no red child.



Case 2.2 Parent (a) is red node.. (a) have black child (b). (b) has red child (c).



Time complexity

- Double red problem doesn't get introduced in deletion
- If we keep on falling in case 1.1, we may have to go towards root
  - └ O(log n) time complexity