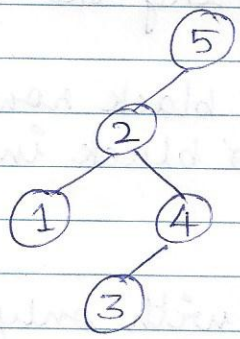
ALGORITHMS- ASSIGNMENT-3

KAVYA SHAMSUNDARA

(1) Is the operation of deletion "commutative" in the sense that deleting x and then y from a binary search tree leaves the same tree as deleting y and then x ? Argue why it is or give a counterexample.

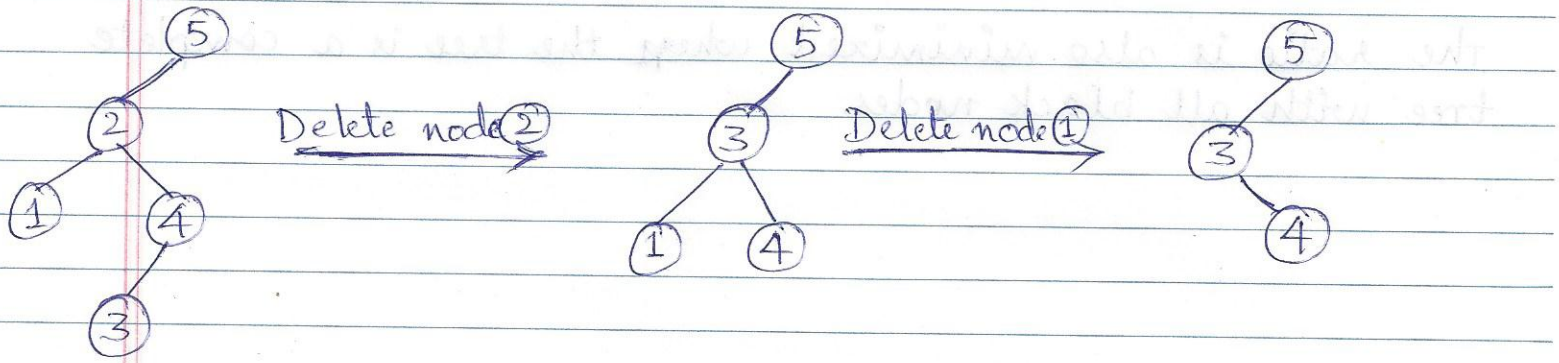
Solution: The deletion operation from a binary search tree is not commutative.

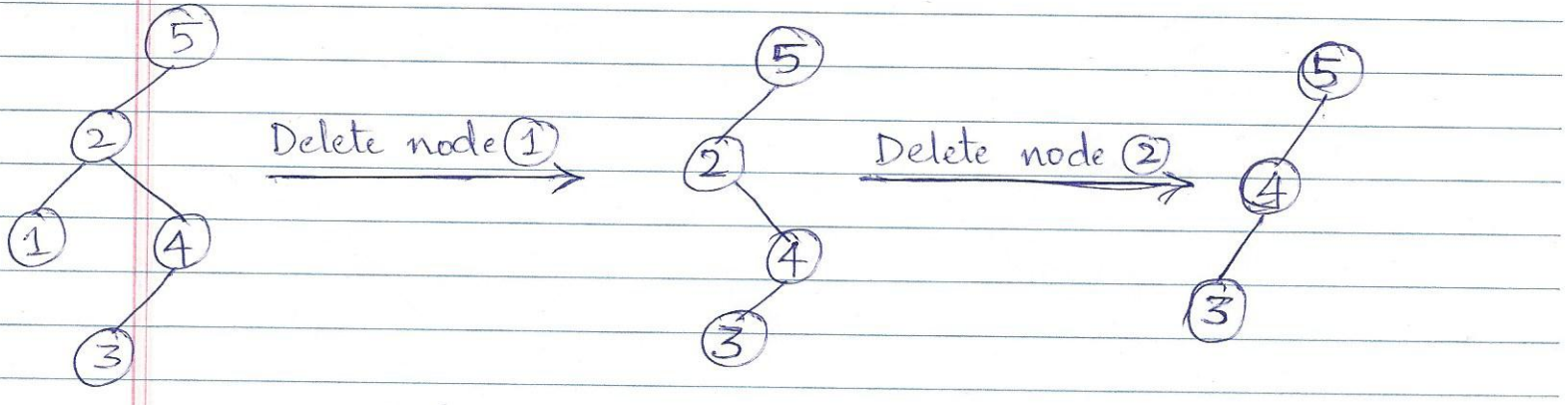
Consider a counter example:



(i) Consider this tree. Let us delete node (2) in first step and then node (1) in second step

(ii) In the next example, consider the same tree. Delete node (1) first & then node (2).





Since the resultant tree is different in both the case, deletion operation is not commutative.

KAVYA SHAMSUNDARA

(2) Describe a red-black tree on n keys that realizes the largest possible ratio of red internal nodes to black internal nodes. What is this ratio? what tree has the smallest possible ratio, & what is the ratio? (2)

Solution:

* The largest possible ratio can be determined if it is a balanced complete binary tree whose red & black nodes alternates in each level.
- The nodes at the bottom are all red ie., ignoring the nil leaf nodes.
- Every red row is double in number of the black row above, therefore, the ratio of red internal nodes to black internal nodes is 2.

* The ratio is smallest when there is a tree with only a root node. The ratio is 0.

The ratio is also minimized when the tree is a complete tree with all black nodes.

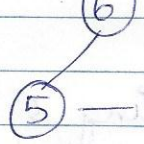
KAVYA SHAMSUNDARA

(3) Consider a red black tree formed by inserting 'n' nodes with RB-INSERT. Argue that if $n \geq 1$, the tree has at least one red node. (3)

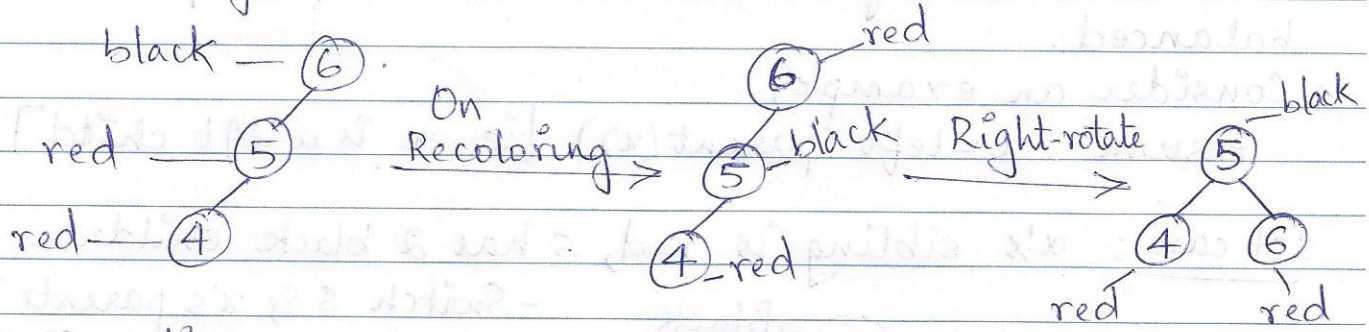
Solution:

* Consider the base case where $n = 2$

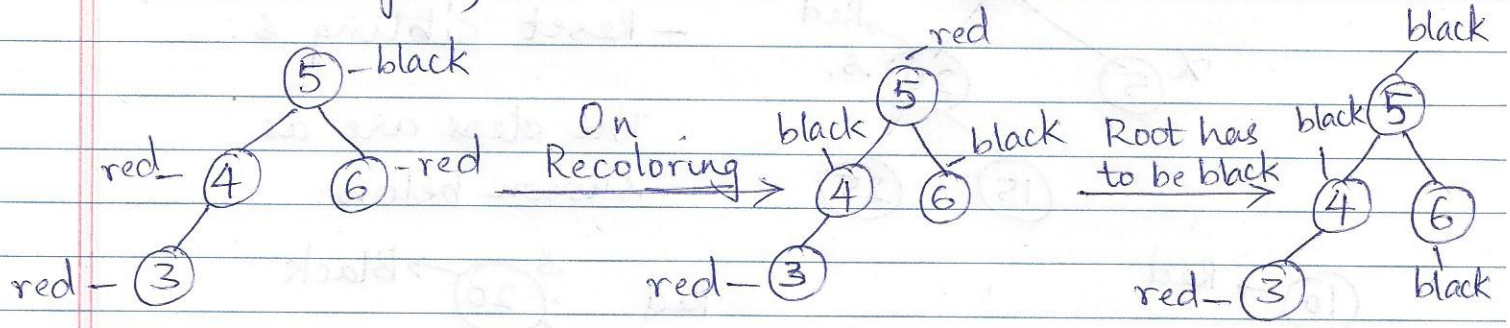
- Let 6 be the root node. (6) — black node
- On inserting 5, the tree becomes:



- On inserting 4,



- On inserting 3,



Therefore, it is proved that for nodes, $n > 1$, the resulting tree has at least one red node.

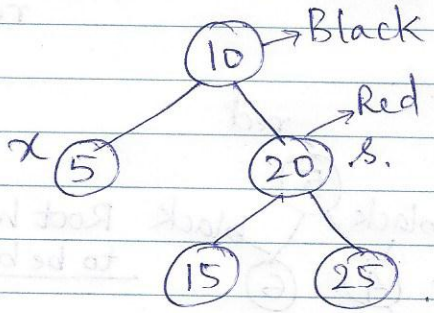
(4) Argue that ~~the~~ after executing RB-DELETE-FIXUP, the root of the tree must be black.

Solution:

- According to the RB-DELETE-FIXUP, the code executes only if the root node is black initially.
- If the root node becomes red, then rotation operation is done to make the black children of the red root node as root node.

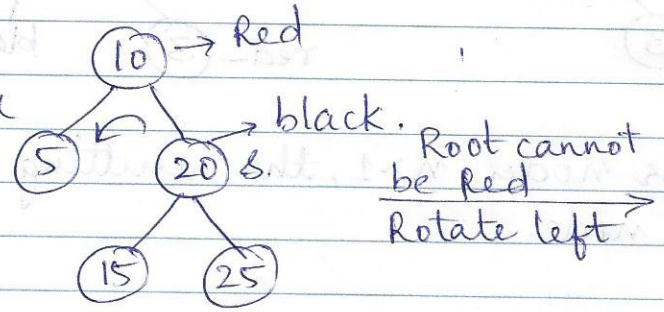
- Consider a node 'x'. x always has a sibling 's'. This is because if x is black & is the child of a deleted black node, then there is a sibling 's' because the tree was previously balanced.
- Consider an example,
  Assume x = left (parent(x)) [ie., x is a left child]

1st case: x's sibling is Red, s has 2 black children.

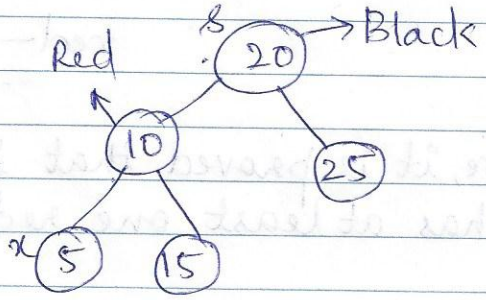


- Switch s & x's parents color
- Rotate parent(x) left
- Reset sibling s.

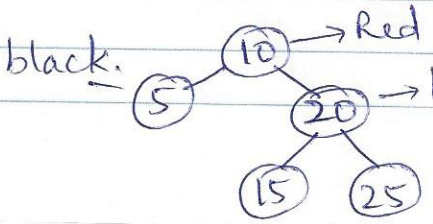
The steps are as shown below.



Root cannot be Red
Rotate left →



2nd case: Consider tree with red root node.



The RB-DELETE-FIXUP will not be executed for the tree mentioned. This is because at the 1st step there is a while loop which checks whether the root is black.

while $x <> root[T]$ and $color[x] = BLACK$.

Hence, for the execution of further statements in RB-DELETE-FIXUP, the root node must be black always.