5) Problem 14.2-2 on Page 347

Homework Assignment 3

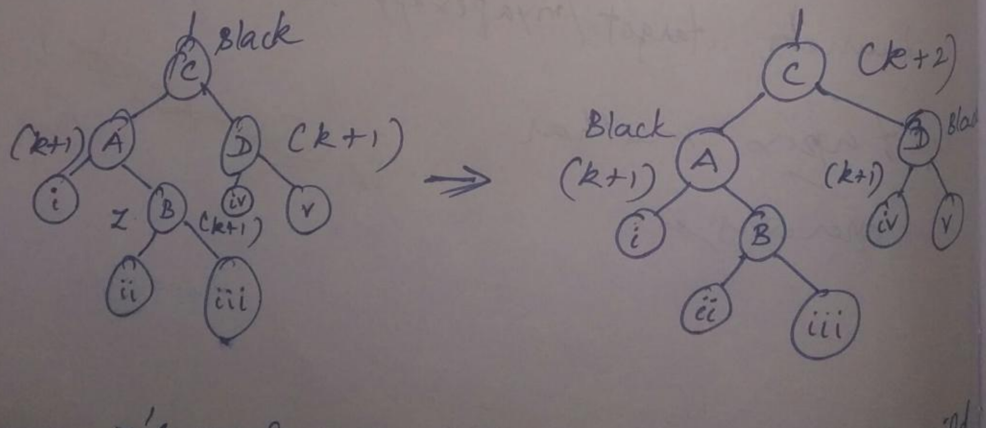
5) Problem 14.2-2 (Page 347)

Solution: By theorem 14.1, it is possible to maintain the black-height. This is because the black height can be computed from information at the node and its two children. It can also be computed from just one childs' Information. (i.e black height of red child / black height of black child plus one)

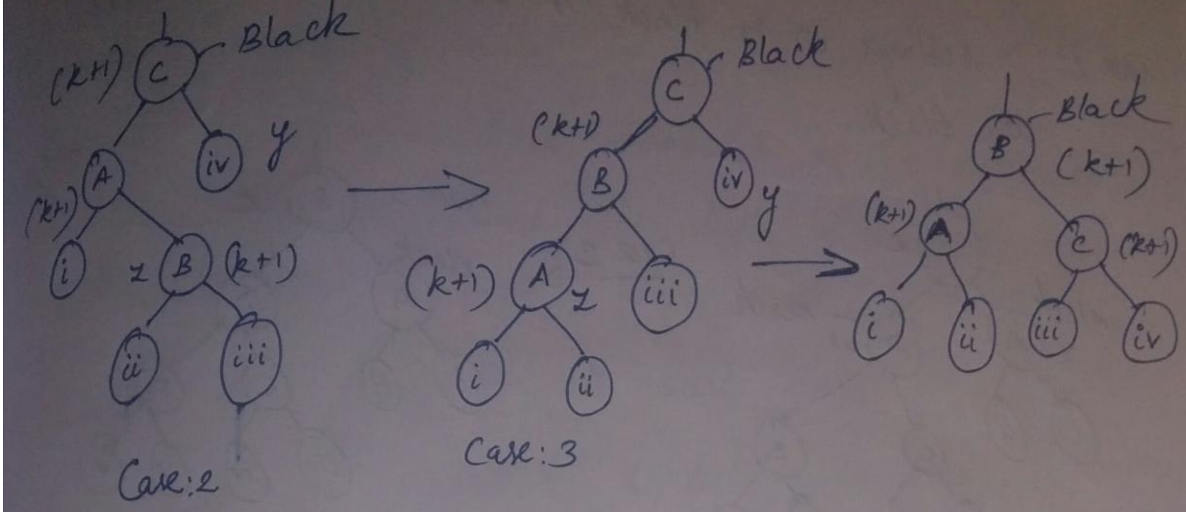
* Consider red black tree operations (RB-INSERT-FIXUP, RB-DELETE-FIXUP)

For RB-INSERT-FIXUP, consider the cases

(i) Case 1: Z's uncle is red.



(ii) Case 2: z's uncle y is black & z is a right child



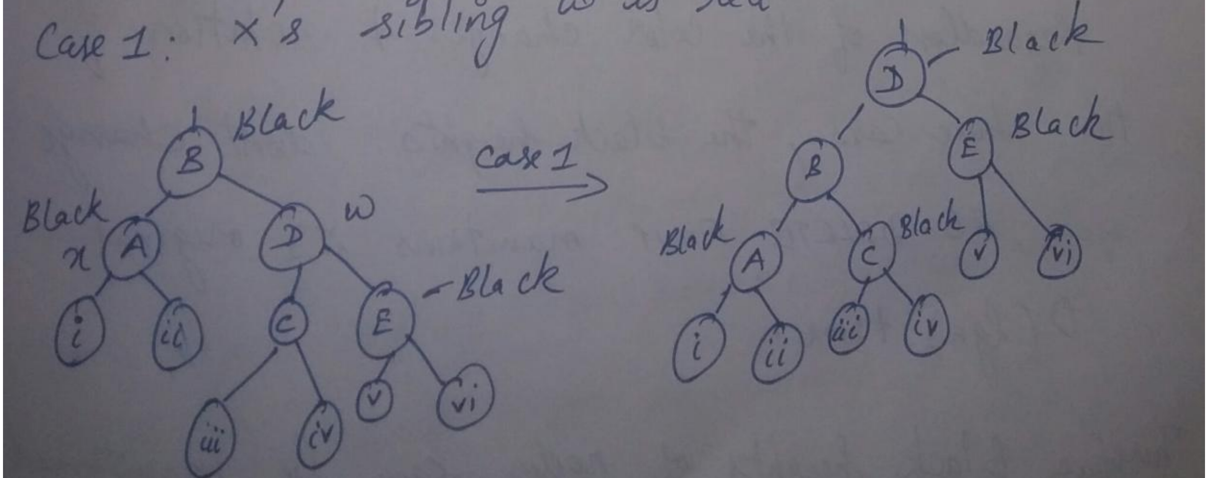
Case:2 Case:3

Cases 3: z's uncle y is black, & z is a left child.

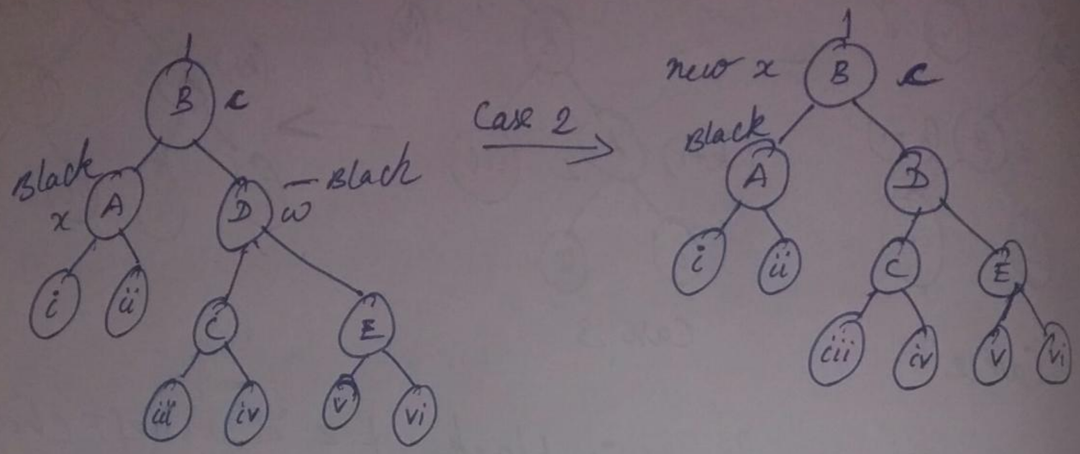
- with subtrees (i), (ii), (iii), (iv), (v) of black height k, we see that even with color changes & rotations the black heights of nodes A, B, C remains the same. (k+1). ∴ The RB-INSERT.FIXUP maintains its original $O(\lg n)$ time.

* Consider RB-Delete-Fixup.

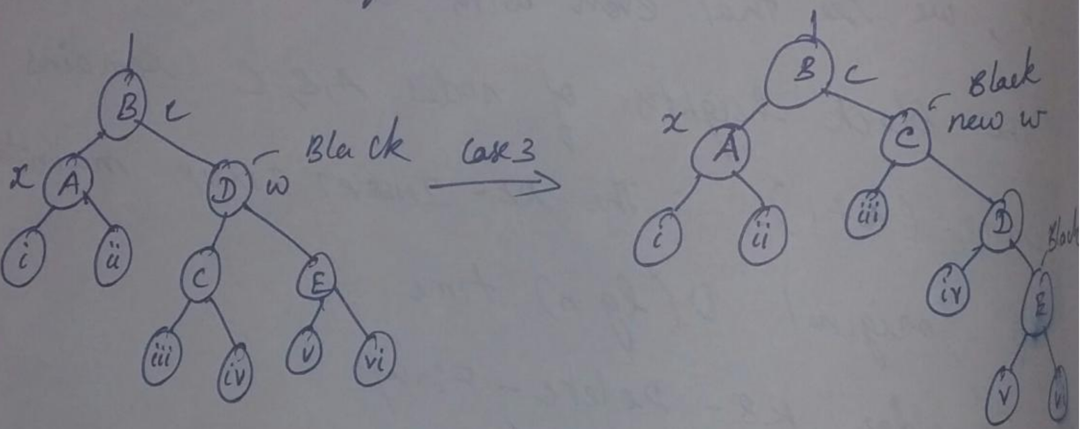
Case 1: x's sibling w is red.



Case 2: x' siblings w is black & both of w's children are black.



Case 3: If x's sibling w is black, w' left child is red, and w's right child is black.



Regardless of the color changes & rotation of the above cases, the black heights dont change.

∴ RB-DELETE FIXUP maintains it's original O(lgn) time.

Therefore Black heights of nodes can be maintained as attributes in the node of the tree without affecting the asymptotic performance of any of the red-black tree operations.

5th problem continuation and 4th problem 13.4-6 on page 330

The red depths of nodes in a red black tree Cannot be maintained as attributes in the node of the tree. This is because, the depth of a node depends on depth of its parent. When depth of the node changes, the depth of all nodes below it in the tree must be updated. Updating the root node, causes other (n-1) nodes to be updated, which means that operations on the tree that change node depths might not run in O(n lg n) time.

4) Problem 13.4-6 on page 330

Solution: Case 1 occurs only if x's sibling w is red. If x.p (Parent of x) were red, then there would be two reds in a row, x.p (which is also w.p (Parent of w)) and w. Therefore we have two reds in a row even before calling RB-DELETE. Thus x.p should be black at the start of case 1.

3rd Problem 13.3-4 on Page 322 and 2nd Problem 13.1-5 on Page 312

3) 13.3-4 Page (322)

* Colors are set to red only in cases 1 and 3, and in both situations ((z.p).p) [parent of (parent of z)] that is reddened. If ((z.p).p) is sentinel, then (z.p) is the root. By Line 1 of RB-Insert-Fixup and part b of Loop Invariant [If z.p is root, then z.p is black], we have dropped out of the Loop in RB-Insert-Fixup The Acute distinction is in Case 2, where we set z ← z.p, before coloring (z.p).p red. As rotation is done before recoloring, the identity [(z.p).p] is same before and after Case 2, there is no problem.

2) 13.1-5 Page (312)

* By Property 5 of Red-Black trees, For each node, all simple paths from the node to descendent leaves contains the same number of Black nodes. therefore by Property 5, the

longest and shortest path must contain the same number of Black nodes. In the Longest path at least every other node is black. In the shortest path, at most every node is black. Since two paths contains equal number of black nodes, the length of longest path is at most twice the length of shortest path.

1) Pg 12.3-2 (Page 299)

*(a) When we insert a node 'n' into the Binary Search tree at the depth h, we have to compare h nodes. when we search for a node 'n' in the Binary Search Tree [assuming the target node is at depth h], we have to compare 'h' times to find the right branch and at last to compare the node at the branch with node 'n'. ∴ The examined nodes are totally h+1.

(b). The nodes (which have children) between root & leaves (Including root) are called Internal nodes. ∴ With n internal nodes, we have (n+1) external nodes. [nodes which have no children]

1st Problem continuation.

∴ for successful search, we have a cost of $\underline{1 + 2I_i}$ comparision (where I_i is the number of comparisions from the root to internal node i). Unsuccessful search will terminate at external nodes at cost of $2E_j$ [where E_j is length of ~~path~~ from root to external node j].

Since $E_j = E_i + 2I$ [E_i is length of path from root to internal node]

Unsuccessful search $= 2(E_j) = 2(E_i + 2I) + 1$ [since n+1 external node]

$= \underline{(2E_i + 4I + 1)}$ Comparisions

In Both successful & unsuccessful search the constant (1), signifies from the fact that it takes a extra comparision to compare the element, compared to the search of element during its insertion into the tree.