Illinois Institute of Technology
Department of Computer Science

Solutions to Homework Assignment 3

CS 430 Introduction to Algorithms
Spring Semester, 2018

Solution:

1. **Problem 12.3-4 on page 299**

   Deletion is not commutative. One counter example is given in Figure 1.

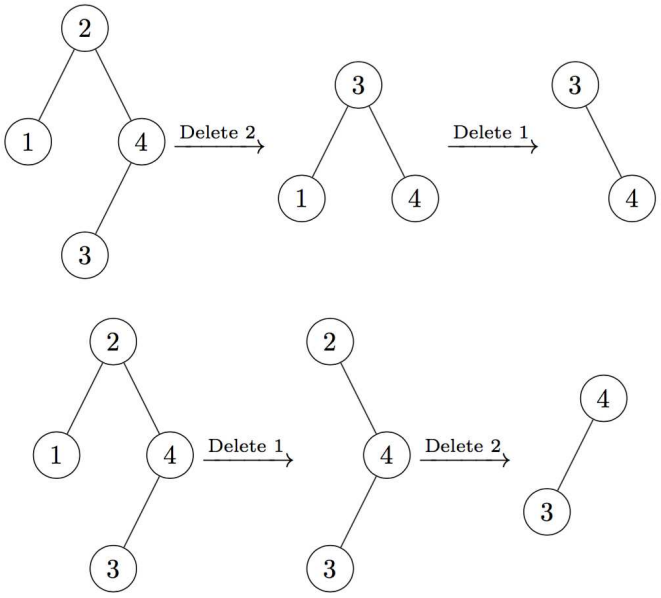


Figure 1: Counter example

2. **Problem 13.1-6 on page 312**

   The case with largest possible number of internal nodes happens when every other node in each path is a black node. This is produced by a complete binary tree which has alternating levels of black and red nodes. Since the black height is k, the height of the tree is $2k$. The total number of internal nodes would be $\sum_{i=0}^{2k-1} 2^i = 2^{2k} - 1$.

   The case with smallest possible number of internal nodes happens when every node is black. That's exactly a complete binary tree with k levels. The total number of internal nodes would be $\sum_{i=0}^{k-1} = 2^k - 1$.

3. **Problem 13.3-4 on page 322**

   Observe that **RB-INSERT-FIXUP** will start from the insertion node z, which is not a $T.nil$ node, and only set node $z.p.p$'s color to RED. Also this procedure will keep going up by level 2 each iteration, until reaches z's parent node $z.p$ as BLACK. So the only chance **RB-INSERT-FIXUP** can set $T.nil$

to red is when $z.p$ is *root* node, in which case $z.p.p$ will be node *T.nil*. But since *root* node is always BLACK, the while condition $z.p.color$==RED ensures when $z.p$ is *root* node, the while loop will halt. So there is no chance *T.nil.color* can be set to RED by **RB-INSERT-FIXUP**.

4. **Problem 13.4-6 on page 330**

   Observe that w is x's sibling, according to the fourth property of the red-black tree, that "*no two red nodes are adjacent. That is, no red node has a red parent or a red child*", if the condition $w.color ==$ RED satisfies, then w's parent, which is $x.p$ must be BLACK. So $x.p$ must be black at the start of the case 1.

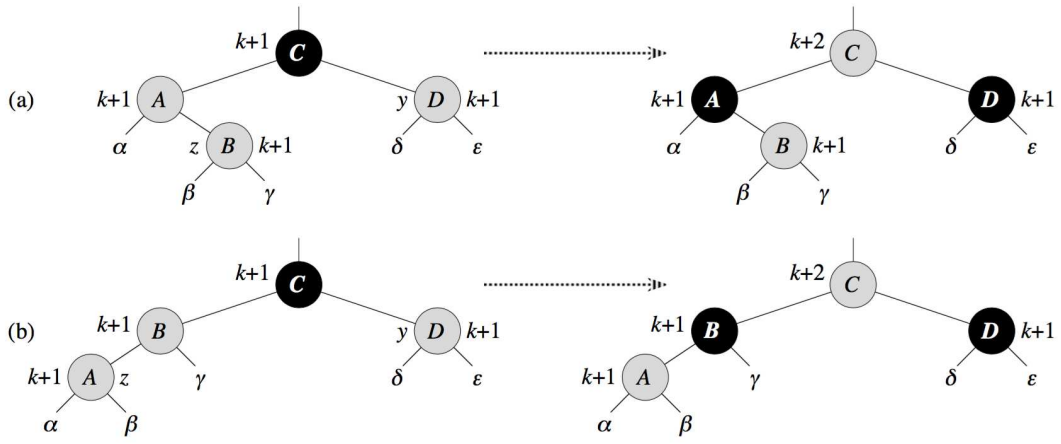
5. **Problem 14.2-2 on page 347**

   Yes, we can maintain black-heights as attributes in the nodes of the tree without affecting the asymptotic performance of the red-black tree operations. According to the theorem 14.1 on page 346, an attribute f can be maintained in all nodes without asymptotically affecting the $O(\lg n)$ performance of the red-black tree, only if the value of f for each node x depends on only the information in nodes x, $x.left$ and $x.right$, possibly including $x.left.f$ and $x.right.f$. Because the black-height of a node can be computed from the information at the node and its two children, so we can maintain the black-heights of nodes in a red-black tree. Actually the black-height of a node can be simply computed from just one child node. The black-height of a node is the black-height of a red child, or the black-height of a black child plus one. We do not need to check the second child node because of the property 5 of red-black tree.

   Let's see how to maintain the black-heights of nodes. Because operation **RB-INSERT** and **RB-DELETE** would probably change node colors, in which potentially cause some nodes' black-height changes, and since **RB-INSERT** and **RB-DELETE** are fixed up by **RB-INSERT-FIXUP** and **RB-DELETE-FIXUP** respectively, we only need to show how to maintain the black-height in all nodes in these two operations.

   We follow the same analysis in CLRS. For **RB-INSERT-FIXUP**, three possible cases need to be considered:

   Case 1: z's uncle y is red:

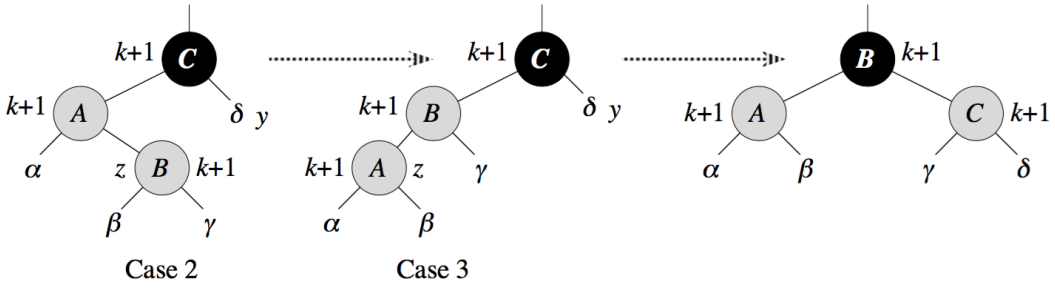


   - Assume before insertion all subtrees $\alpha, \beta, \gamma, \delta, \epsilon$ have the same black-height k with a black root, so the nodes A, B, C, D all have the black height $k + 1$.

- After setting node C's color to red and B, D's color to black, only node C's black height changes, increased by 1. So we only need to set $z.p.p.bh = z.p.p.bh + 1$ (bh is notated as black-height).
- Since the number of black nodes between z and $z.p.p$ do not change, black height of nodes above $z.p.p$ will not be affected by this color change.

Case 2: z's uncle y is black and z is a right child:

Case 3: z's uncle y is black and z is a left child:

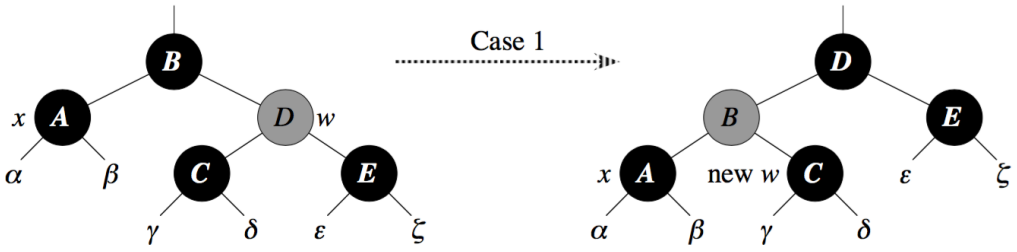


- After rotation and color change, node A, C's black height will remain the same due to the unchanged black height k of subtree $\alpha, \beta, \gamma, \delta$. Node B's black height will still be $k + 1$ considering its two new children A, C's black height.
- So there is no need to modify any node's black height in these two cases.

Based on the analysis, we can maintain its original $O(\lg n)$ time for **RB-INSERT** operation even we add black height attribute to each node in the red-black tree.

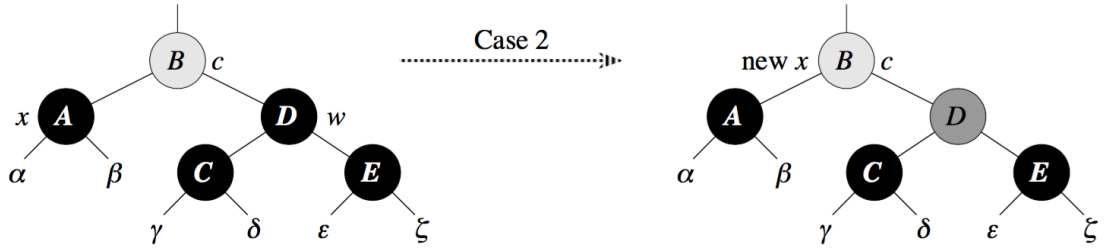
Now let's see how to maintain the property in **RB-DELETE-FIXUP** operation. There would be 4 cases need to be considered.

Case 1: x's sibling w is red:



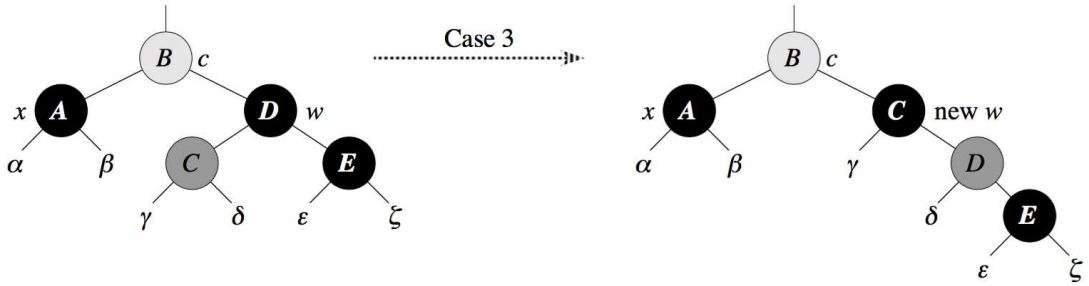
- Because subtree $\alpha, \beta, \gamma, \delta, \epsilon, \xi$'s black height does not change, so node A, C, E's black height remains the same.
- Originally $B.bh = A.bh + 1$ and $D.bh = E.bh + 1$, after rotation and color change, $B.bh = A.bh + 1$ and $D.bh = E.bh + 1$, while $A.bh$ and $E.bh$ remain unchanged, so does the black height of node B and D.
- So in this case, it only changes the structure of tree while does not affect the black height of each node. Nothing need to be changed here.

Case 2:** x**'s sibling** w **is black, and both of** w**'s children are black:



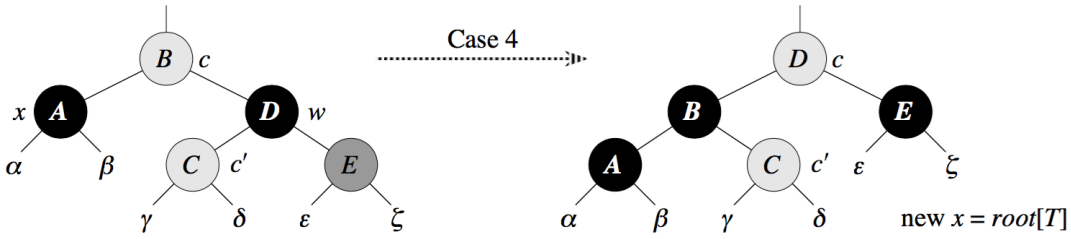
- In this case, w's color is set to red and x is moved to its parent node, along with its "extra" black.
- We only need to update B's black height here, with the value of x's black height, that's to say $x.p.bh = x.bh$.

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



- The analysis is the same with case 1. It only changes the tree structure without affecting each node's black height. We do not need to update node's black height here.

Case 4:** x**'s sibling** w **is black, and** w**'s right child is red:



- Node A, C, E's black height keep unchanged, due to their unchanged subtree.
- Node B, D's black height is potentially changed. We need to set $x.p.bh = x.bh + 1$ and $x.p.p.bh = x.p.bh + 1$. This is implemented after line 20 in **RB-DELETE-FIXUP**.

Thus we can see for both insertion and deletion, we can maintain the black height attribute of each node without affecting the asymptotic performance of red-black operations.

No, we cannot maintain the depths of nodes without affecting the asymptotic performance of red-black tree operations, because the depth of a node depends on the depth of its parent. We cannot calculate a node's depth simply from the information of its children nodes. When the depth of a node changes, the depths of all nodes below it in the tree rooted by this node must be updated. Updating the root node causes $n-1$ other nodes to be updated, which means that the operations on the tree that changes node depths might not run in $O(\lg n)$ time.