Illinois Institute of Technology
Department of Computer Science

Solutions to Homework Assignment 3

CS 430 Introduction to Algorithms
Spring Semester, 2016

Solution:

1. For inserting a node at level l, all the nodes in the path till its supposed parent at level $l-1$ are examined, so $l-1$ comparisons are made; for searching the node at level l, number of nodes examined also includes the node at level l which is searched for. Therefore, we examined one more node when searching the value.

   Similarly, number of comparisons made in the unsuccessful search of node x is one less than that in the successful search reflecting that x is in the tree, because one more comparison is made at x.

2. In the longest path, at least every other node is black. In the shortest path, at most every node is black. Since the two paths contain equal numbers of black nodes, the length of the longest path is at most twice the length of the shortest path.

   We can say this more precisely, as follows: Since every path contains $bh(x)$ black nodes, even the shortest path from x to a descendant leaf has length at least $bh(x)$. By definition, the longest path from x to a descendant leaf has length $height(x)$. Since the longest path has $bh(x)$ black nodes and at least half the nodes on the longest path are black (by property 4), $bh(x) \geq height(x)/2$, so length of longest $path = height(x) \leq 2 \cdot bh(x) \leq$ twice length of shortest path .

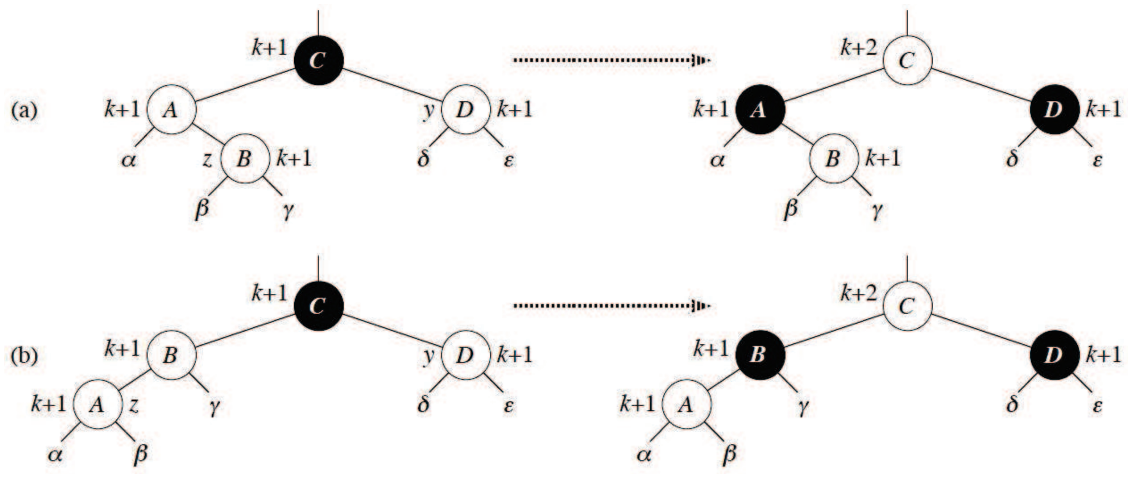
3. Colors are set to red only in cases 1 and 3, and in both situations, it is $z.p.p$ that is reddened. If $z.p.p$ is the sentinel, then $z.p$ is the root. By part (b) of the loop invariant and line 1 of RB-INSERT-FIXUP, if $z.p$ is the root, then we have dropped out of the loop. The only subtlety is in case 2, where we set $z \leftarrow z.p$ before coloring $z.p.p$ red. Because we rotate before the recoloring, the identity of $z.p.p$ is the same before and after case 2, so there's no problem.

4. Case 1 occurs only if x's sibling w is red. If $x.p$ were red, then there would be two reds in a row, namely $x.p$ (which is also $w.p$) and w, and we would have had these two reds in a row even before calling RB-DELETE.

5. Yes, by Theorem 14.1, because the black-height of a node can be computed from the information at the node and its two children. Actually, the black-height can be computed from just one child's information: the black-height of a node is the black-height of a red child, or the black height of a black child plus one. The second child does not need to be checked because of property 5 of red-black trees. Within the RB-INSERT-FIXUP and RB-DELETE-FIXUP procedures are color changes, each of which potentially cause $O(\lg n)$ black-height changes. Let us show that the color changes of the fixup procedures cause only local black-height changes and thus are constant-time operations. Assume that the black-height of each node x is kept in the field $bh[x]$. For RB-INSERT-FIXUP, there are 3 cases to examine.

   Case 1: z's uncle is red

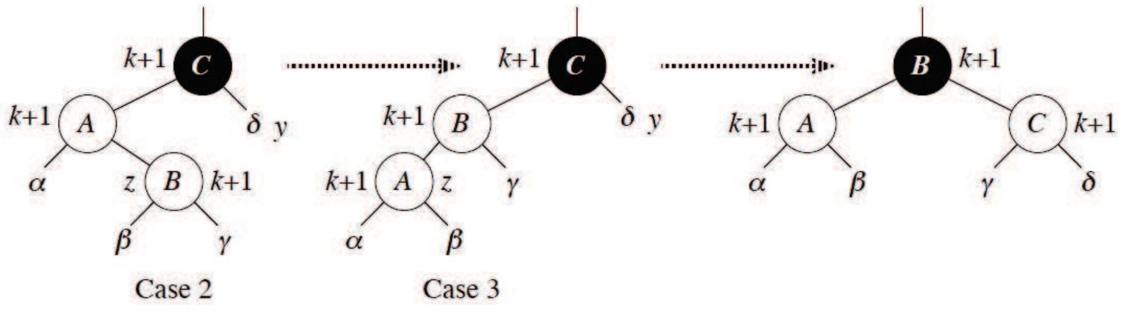
   - Before color changes, suppose that all subtrees $\alpha, \beta, \gamma, \delta, \epsilon$ have the same black-height k with a black root, so that nodes A, B, C, and D have blackheights of $k+1$.



- After color changes, the only node whose black-height changed is node C. To fix that, add $bh[z.p.p] = bh[z.p.p] + 1$ after line 7 in RB-INSERTFIXUP.
- Since the number of black nodes between $z.p.p$ and z remains the same, nodes above $z.p.p$ are not affected by the 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.



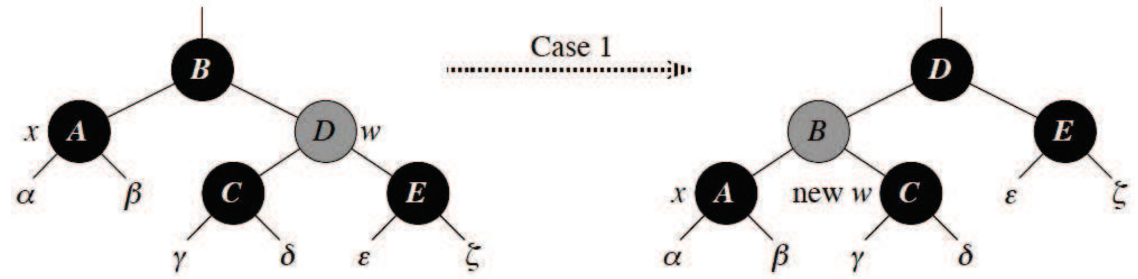
- With subtrees $\alpha, \beta, \gamma, \delta, \epsilon$ of black-height k, we see that even with color changes and rotations, the black-heights of nodes A, B, and C remain the same $(k + 1)$.

Thus, RB-INSERT-FIXUP maintains its original $O(\lg n)$ time.

For RB-DELETE-FIXUP, there are 4 cases to examine.

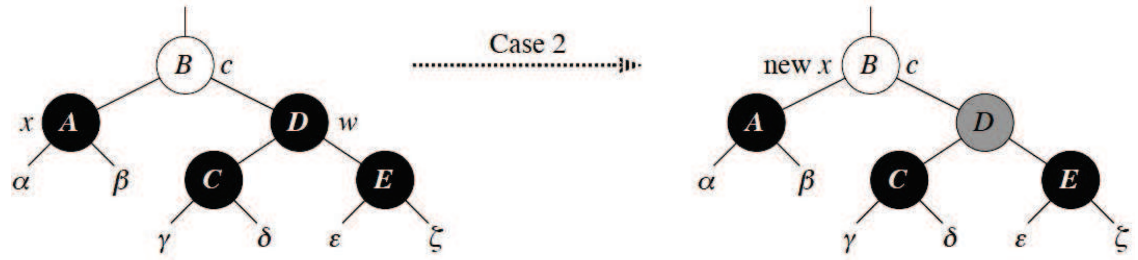
Case 1: x's sibling w is red.

- Even though case 1 changes colors of nodes and does a rotation, blackheights are not changed.



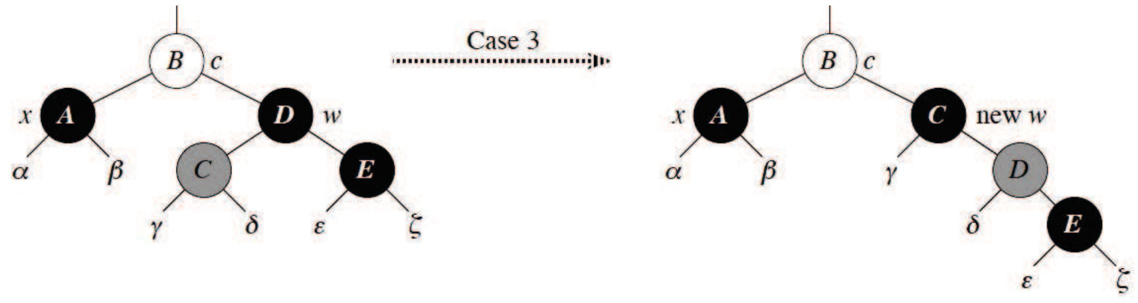
- Case 1 changes the structure of the tree, but waits for cases 2, 3, and 4 to deal with the "extra black" on x.

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



- w is colored red, and x's "extra" black is moved up to $x.p$.
- Now we can add $bh[x.p] = bh[x]$ after line 10 in RB-DELETE-FIXUP.
- This is a constant-time update. Then, keep looping to deal with the extra black on $x.p$.

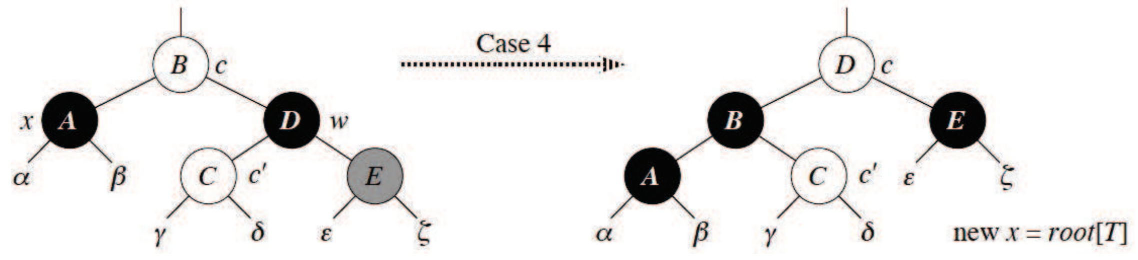
Case 3: x's sibling w is black, w's left child is red, and w's right child is black.



- Regardless of the color changes and rotation of this case, the black-heights don't change.

- Case 3 just sets up the structure of the tree, so it can fall correctly into case 4.

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



- Nodes A, C, and E keep the same subtrees, so their black-heights don't change.
- Add these two constant-time assignments in RB-DELETE-FIXUP after line 20: $bh[x.p] = bh[x]+1$. and $bh[x.pp] = bh[x.p] + 1$.
- The extra black is taken care of. Loop terminates.

Thus, RB-DELETE-FIXUP maintains its original $O(\lg n)$ time.

Therefore, we conclude that black-heights of nodes can be maintained as fields in red-black trees without affecting the asymptotic performance of red-black tree operations.