

## 1 Insertion in red-black trees

Insertion of a node  $z$  in an r-b tree is similar to insertion in a BST tree.

$z$  **becomes the root**. If node  $z$  is inserted into an empty tree, we color  $z$  BLACK, and make  $z$  the root of the tree. Otherwise, the tree is not empty and,

$z$  is **not the root**. We perform the standard BST-Insert operations and color  $z$  red.

**Possible Problem.** When we color  $z$  red, if the parent  $p_z$  of  $z$  is also red, we have a problem. Note that in that case the grandparent  $g_z$  of  $z$  must be black (or  $p_z$  could not have been red). Towards this we need to apply a function  $FIX(z)$  recursively to fix the RED color of  $z$ . (If  $z$  becomes the root, fixing the root is straightforward!)

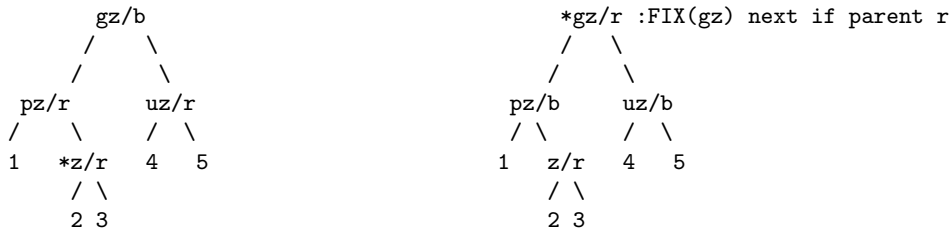
**Case Description.** Call cases  $XYc$ .  $X$  is based on the  $p_z:g_z$ ,  $Y$  on the  $z:p_z$  and  $c$  on the  $color(uz)$  relationship, where  $uz$  is the sibling of  $p_z$ . Thus  $X, Y$  can be R or L and  $c$  can be r or b.

### 1.1 Case 1: LLr, LRr and RRr, RLr

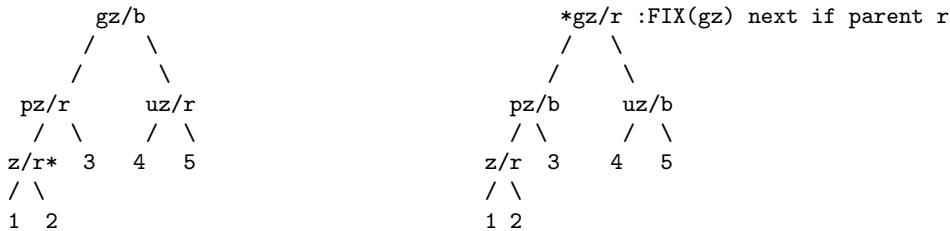
**Case 1.** The first case involves the Insertion subcases **LRr and LLr** which are shown. Cases **RRr and RLr** are not shown but are symmetric. These cases require node recolorings only. Note that if  $g_z$  is the root its color cannot change; this causes an increase to the blackheight of descendant nodes.  $FIX$  may cause a total of  $O(\lg n)$  recursive calls higher in the tree.

// LRr, LLr shown (RRr, RLr symmetric and not shown)

Case 1a: LRr ::  $g_z$  b  $\rightarrow$  r;  $g_z$  and  $p(g_z)$  may become red; Call  $Fix(g_z)$  next.



Case 1b: LLr :: Same as before ( $g_z$ : b  $\rightarrow$  r)

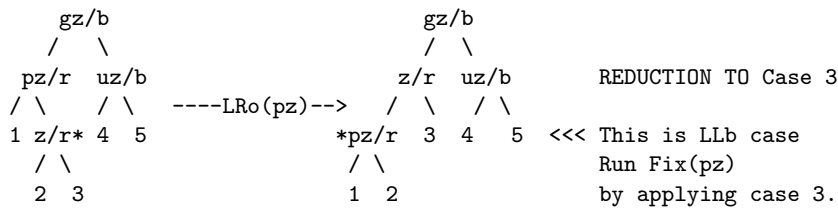


## 1.2 Case 2: LRb, and Case 3: LLb

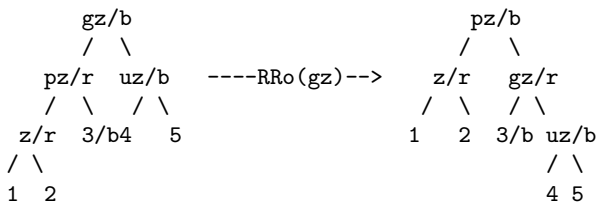
Case 2 covers the case of LRb, and Case 3 the case of LLb. Case 2 is ALWAYS FOLLOWED by Case 3. RLb and RRb are symmetric(not shown).

\*\*\*\*\* A star (\*) shows the node on which FIX is run.

Case 2:LRb is reduced to Case 3:LLb and resolved.



Case 3:LLb



After the single rotation is performed there can be no way that there are two consecutive RED nodes in a path from the root to pz (root of the subtree in Case 3). Therefore after a single rotation we are done.

**Conclusion.** Insertion requires  $O(\lg n)$  recolorings (Case 1) and  $O(1)$  rotations (Cases 2 and 3).

## 2 Deletion in red-black trees

Deletion in an r-b tree is similar to Deletion in a BST tree. When we perform Delete( $z$ ), a node is spliced out; this node is called  $x$ . If  $z$  has no or one child, then  $x$  is  $z$  otherwise  $x$  is the successor (or the predecessor) of  $z$ . In any of these three cases we call  $y$  the only child of  $x$  (if  $x$  has no children, then  $y$  is the NULL only child of  $x$  reminding ourselves that in an rb-tree there is only one NULL node), and  $py$  the new parent of  $y$  after the spliceout, which was previously the parent of  $x$ .

- a.  $(x, y, py) = (r, b, b)$ . If  $x$  is red, then  $y$  must be black and  $p(y)$  must be black or otherwise  $x$  should have been black. Splicing out  $x$  causes no violations whatsoever.
- b.  $(x, y, py) = (b, ?, ?)$  Splicing out a black  $x$  causes an RB3 violation; subcases are as follows.
  - b1.  $(x, y, py) = (b, r, ?)$ . If  $y$  is red, we recolor  $y$  black and the violation is resolved.
  - b2.  $(x, y, py) = (b, b, ?)$  and  $y$  is root. If  $y$  is black and becomes the root, no RB3 violation occurs, because all the paths from the root  $y$  will have black height one less.
  - b3.  $(x, y, py) = (b, b, ?)$  and  $y$  not root. If  $y$  is black but not the root, we have a violation of RB3 that can not be resolved immediately. We “transfer” the BLACK color of  $x$  to  $y$  by coloring  $y$  DOUBLE-BLACK. We then need to fix  $y$  by calling FIXDELETE( $y$ ), i.e. a node  $y$  of FIXDELETE is a node “colored” BLACK twice.

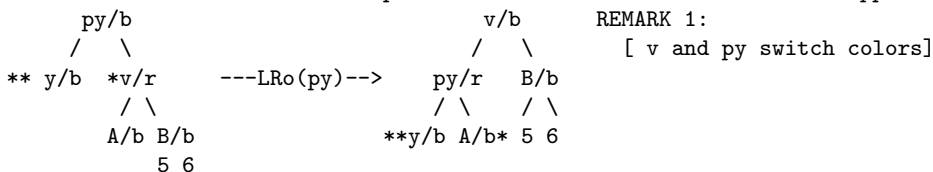
**Reminder: delete( $z$ )  $\rightarrow$  spliceout( $x$ )  $\rightarrow$  FixDelete( $y$ ).**

**Case Description.** All cases are labeled  $Xcn$ , where  $X$  denotes the orientation of  $y:py$ ,  $c$  the color of the sibling  $u$  of  $y$  and  $n$  the number of red children of  $u$  (i.e. sibling of  $y$ ).

### 2.1 Cases 1 and 2

Comment:  $Lr1$  and  $Lr2$  are not possible; a red node CANNOT have red children.

Case 1:  $Lr0$  : A left rotation is performed and then Case 2a or 3 or 4 applies.



Case 2.  $Lb0$

Subcase 2a. If  $py$  is  $r$  (Case 1) color  $py$  with  $b$  and color  $v$  with  $r$  and stop

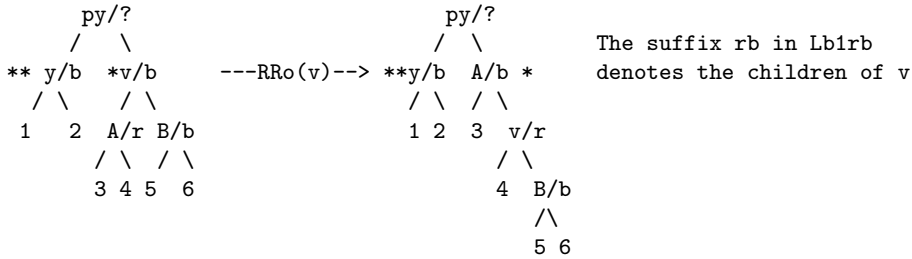


Subcase 2b, If  $py$  is  $b$  FIXDELETE( $py$ )  $py$  plays the role of  $y$  and  $py$  also carries the extra black inherited by  $y$  due to the  $x$  splice out.

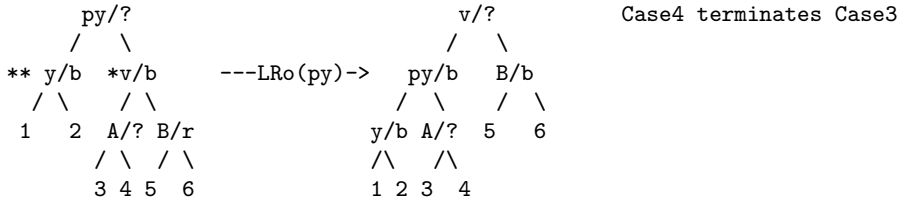


## 2.2 Cases 3 and 4

Case 3: Lb1rb : Is transformed into case 4 immediately A,v exchanging colors.



Case 4: Lb2 and Lb1br (there is neither Lb1bb=Lb0 neither Lb1rr=Lb2) Case4 terminates Case3



Running time is  $O(\lg n)$  as well. Cases 1, 2a, 3, 4 terminate in  $O(1)$  time, Case 2b advances (moves towards the top) one level every time it is executed, and the height of the RB tree is  $O(\lg n)$ .