RB-INSERT-FIXUP(*T*, *z*)

1. **while** *z.p.color* == RED
2. **if** z.p == z.p.p.left
3. y = *z.p.p.right*
4. **if** *y.color* == RED
5. *z.p.color* = BLACK //case1
6. *y.color* = BLACK //case1
7. *z.p.p.color* = RED //case1
8. *z* = *z.p.p* //case1
9. **else**
10. **if** *z* == *z.p.right* //case2
11. *z* = *z.p* //case2
12. LEFT-ROTATE(*T*, *z*) //case2
13. *z.p.color* = BLACK //case3
14. *z.p.p.color* = RED //case3
15. RIGHT-ROTATE(*T*, *z.p.p*) //case3
16. else
17. *y* = *z.p.p.left*
18. **if** *y.color* == RED
19. *z.p.color* = BLACK //case4
20. *y.color* = BLACK //case4
21. *z.p.p.color* = RED //case4
22. *z* = *z.p.p* //case4
23. **else**
24. if *z* == *z.p.left* //case5
25. *z* = *z.p* //case5
26. RIGHT-ROTATE(*T*, *z*) //case5
27. *z.p.color* = BLACK //case6
28. *z.p.p.color* = RED //case6
29. LEFT-ROTATE(*T*, *z.p.p*) //case6
30. T.root.color = BLACK

RB-DELETE-FIXUP(*T*, *x*)

1. **while** *x* ! = *T.root* and *x.color* == BLACK
2. **if** *x* == *x.p.left*
3. *w* = *x.p.right*
4. **if** *w.color* == RED
5. *w.color* = BLACK
6. *x.p.color* = RED //case1
7. LEFT-ROTATE(*T*, *x.p*) //case1
8. *w* = *x.p.right* //case1
9. **if** *w.left.color* == BLACK and *w.right.color* == BLACK
10. *w.color* = RED //case2
11. *x* = *x.p* //case2
12. **else**
13. **if** *w.right.color* == BLACK
14. *w.left.color* = BLACK //case3
15. *w.color* = RED //case3
16. RIGHT-ROTATE(*T*, *w*) //case3
17. *w* = *x.p.right* //case3
18. *w.color* = *x.p.color* //case4
19. *x.p.color* = BLACK //case4
20. *w.right.color* = BLACK //case4
21. LEFT-ROTATE(T, x.p) //case4
22. *x* = *T.root* //case4
23. **else**
24. *w* = *x.p.left*
25. **if** *w.color* == RED
26. *w.color* = BLACK
27. *x.p.color* = RED //case5
28. RIGHT-ROTATE(*T*, *x.p*) //case5
29. *w* = *x.p.left* //case5
30. **if** *w.right.color* == BLACK and *w.left.color* == BLACK
31. *w.color* = RED //case6
32. *x* = *x.p* //case6
33. **else**
34. **if** *w.left.color* == BLACK
35. *w.right.color* = BLACK //case7
36. *w.color* = RED //case7
37. LEFT-ROTATE(*T*, *w*) //case7
38. *w* = *x.p.left* //case7
39. *w.color* = *x.p.color* //case8
40. *x.p.color* = BLACK //case8
41. *w.left.color* = BLACK //case8
42. RIGHT-ROTATE(*T*, *x.p*) //case8
43. *x* = *T.root* //case8
44. *x.color* = BLACK

//迭代写法

PERSISTENT-TREE-INSERT(*T*, *k*)

1. *z* = MAKE-NEW-NODE(*k*)
2. *new-root* = COPY-NODE(*T.root*)
3. *y* = NIL
4. x = new-root
5. **while** *x* != NIL
6. *y* = *x*
7. **if** *z.key* < *x.key*
8. *x* = COPY-NODE(*x.left*)
9. *y.left* = *x*
10. else
11. *x* = COPY-NODE(*x.right*)
12. *y.right* = *x*
13. **if** *y* == NIL
14. new-root = z
15. **elseif** *z.key* < *y.key*
16. *y.left* = *z*
17. else
18. y.right = z
19. **return** new-root

//递归写法

PERSISTENT-TREE-INSERT(*r*, *k*)

1. **if** *r* == NIL
2. *x* = MAKE-NEW-NODE(*k*)
3. **else** *x* = COPY-NODE(*r*)
4. **if** *k* < *r.key*
5. *x.left* = PERSISTENT-TREE-INSERT(*r.left*, *k*)
6. **else**
7. *x.right* = PERSISTENT-TREE-INSERT(*r.right*, *k*)
8. **return** *x*

BALANCE(*x*)

1. **if** |x.left.h - x.right.h| <= 1
2. **return** *x*
3. **elseif** *x.left.h* > *x.right.h*
4. *y* = *x.left*
5. **if** *y.left.h* > *y.right.h*
6. LEFT-ROTATE(*T*, *y*)
7. **return** RIGHT-ROTATE(*T*, *x*)
8. **else**
9. *y* = *x.right*
10. **if** *y.left.h* > *y.right.h*
11. RIGHT-ROTATE(*T*, *y*)
12. **return** LEFT-ROTATE(*T*, *x*)

AVL-INSERT(*x*, *z*)

1. **if** *x* = NIL
2. *z.h* = 0
3. **return** *z*
4. **if** *z.key* <= *x.key*
5. *y* = INSERT(*x.left*, *z*)
6. *x.left* = *y*
7. *y.p* = *x*
8. *x.h* = *y.h* + 1
9. **else**
10. *y* = INSERT(*x.right*, *z*)
11. *x.right* = *y*
12. *y.p* = *x*
13. *x.h* = *y.h* + 1
14. *x* = BALANCE(*x*)
15. **return** *x*