Red-Black Trees

CS130A Coakley

Recall

AVL Trees have properties to enforce balance

- The heights of children can differ by at most 1
- Height of a tree = 1 + max(height-left, height-right)
- Height of NULL is -1

AVL Trees store their height at each node

Other Self-Balancing Trees

We want to keep O(log n) bounds

We are willing to be even more unbalanced than AVL

The maximum depth of a leaf node must still be O(log n) to keep our bounds

We will limit our height to 2 log(N+1)

As an aside, these will have additional computational complexity benefits (sounds better than “theoretical benefits”)

Red-Black Tree Prelude

Your book provides an advanced (top-down) description in Chapter 12, but the textual description is a little lacking.

We will mimic Wikipedia's labelling.

Similar visualizer to AVL (slightly better):

https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

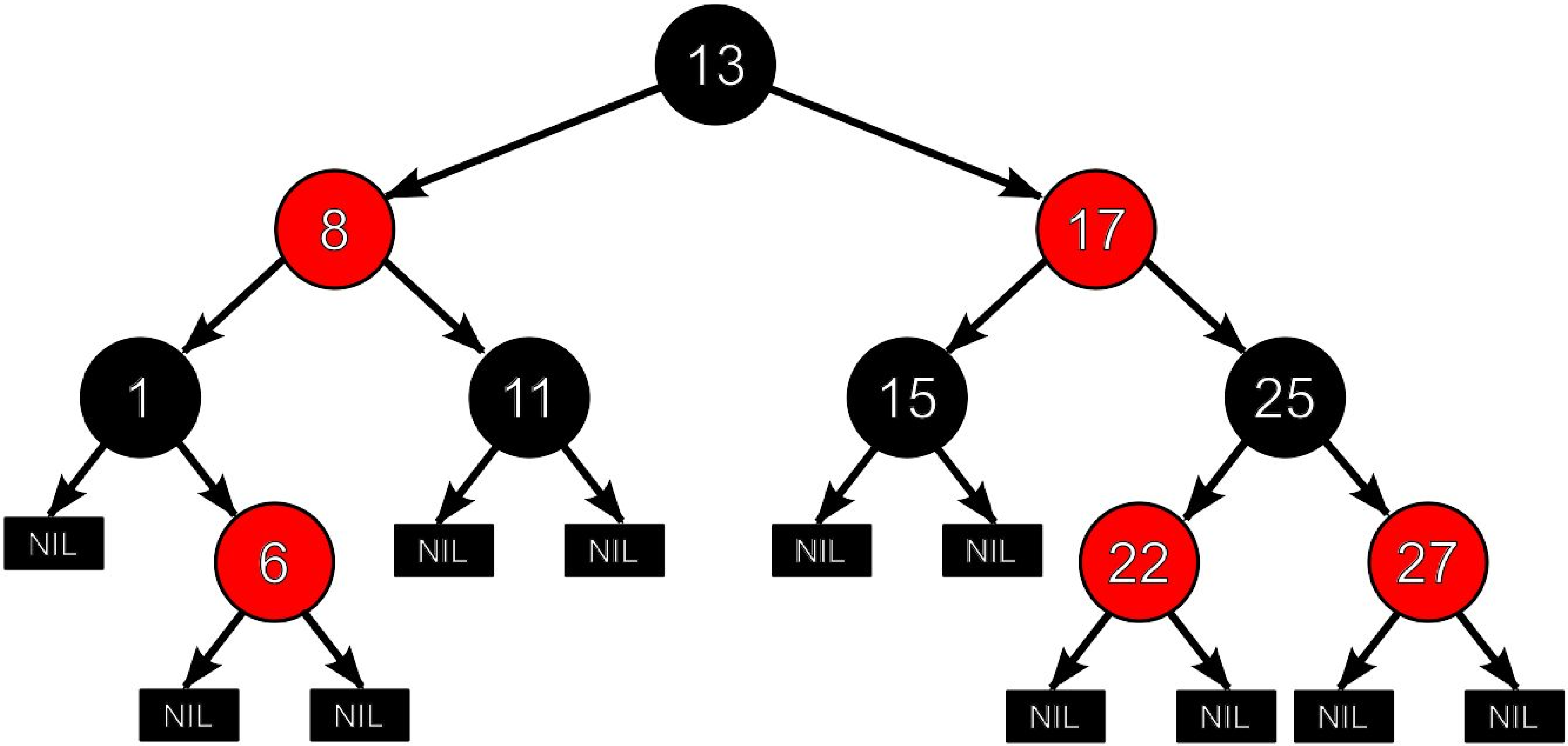
Red-Black Trees

Properties:

1. All nodes are red or black
2. The root is black
3. All leaves (NULL) are black
4. If a node is red, its children must be black
5. The path from a node to all of its leaves contains the same number of black nodes (this gives us a black-height of the tree and defines the black-depth of a node)

The maximum height is 2 log (N+1)

Example



13
8
17
1
11
15
25
NIL
6
NIL
NIL
NIL
NIL
22
27
NIL
NIL
NIL
NIL
NIL
NIL

The maximum height is 2 log (n+1)

bh(v) = black-height of v (excludes v even if black)

h(v) = height of v

Lemma: A subtree rooted at node v has at least $2^{bh(v)}$-1 nodes

Note: This is by induction on height.

Basis: h(v)=0 for NULL gives $2^{bh(v)}-1 = 2^0-1 = 1-1 = 0$

Inductive step: v such that h(v) = k, has at least $2^{bh(v)} - 1$ internal nodes implies that v′ such that h(v′) = k+1 has at least $2^{bh(v')} - 1$ internal nodes.

Since v′ has h(v′) > 0 it is an internal node.

As such it has two children each of which have a black-height of either bh(v′) or bh(v′)-1 (depending on whether the child is red or black, respectively).

By the inductive hypothesis each child has at least $2^{bh(v')} - 1$ internal nodes, so v′ has at least:

$$(2^{bh(v')-1} - 1) + (2^{bh(v')-1} - 1) + 1 = 2^{bh(v')} - 1$$

internal nodes.

Height Bounds Via Lemma

Property 4 (red children are black) guarantees us that at least half of the nodes on any path from the root to to a leaf are black.

The bh(root) of a tree of n nodes is therefore at least h(root)/2-1. Using the lemma,

$n >= 2^{h(root)/2}-1$

$log(n+1) >= h(root)/2$

$h(root) <= 2 log(n+1)$

Operations

The red-black tree is still a binary search tree

Search is O(log n) based on height limit

Insertion and Deletion are special, and involve color changes and rotations.

Insertion

We always insert a red node. It replaces a black NULL-leaf with itself and 2 black NULL-leaves.

Recall the properties:

1. All nodes are red or black
2. The root is black
3. All leaves (NULL) are black
4. If a node is red, its children must be black
5. The path from a node to all of its leaves contains the same number of black nodes (this gives us a black-height of the tree and defines the black-depth of a node)

Some Labels

N is the current node. It is the new node in the base case, but we have to recurse to fix the tree.

P is the parent node

S is the sibling

G is the grandparent

U is the uncle - the parent's sibling, if it exists

Enumerate the Cases

1. N is the root node
2. P is black
3. P and U are red
4. N is left-right or right-left of G, P is red, U is black
5. N is left-left or right-right of G, P is red, U is black

- This is similar to the special cases of the AVL tree, with an additional non-trivial case based on parent color.
- The black-height property will be preserved without needing to store black-height at each node.
- How often do case 4 and 5 occur on the non-recursive call?

N is the Root Node

Change the color to black to satisfy property 2

P is Black

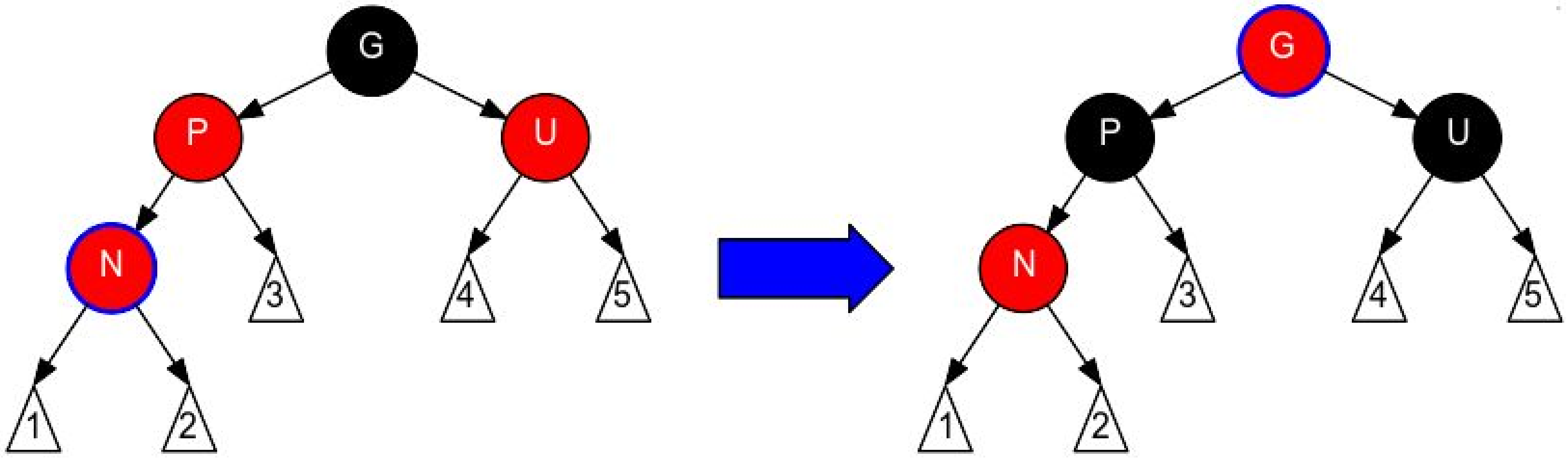
Done

P and U Are Red

Change P and U to black, Change G to red, recurse on G

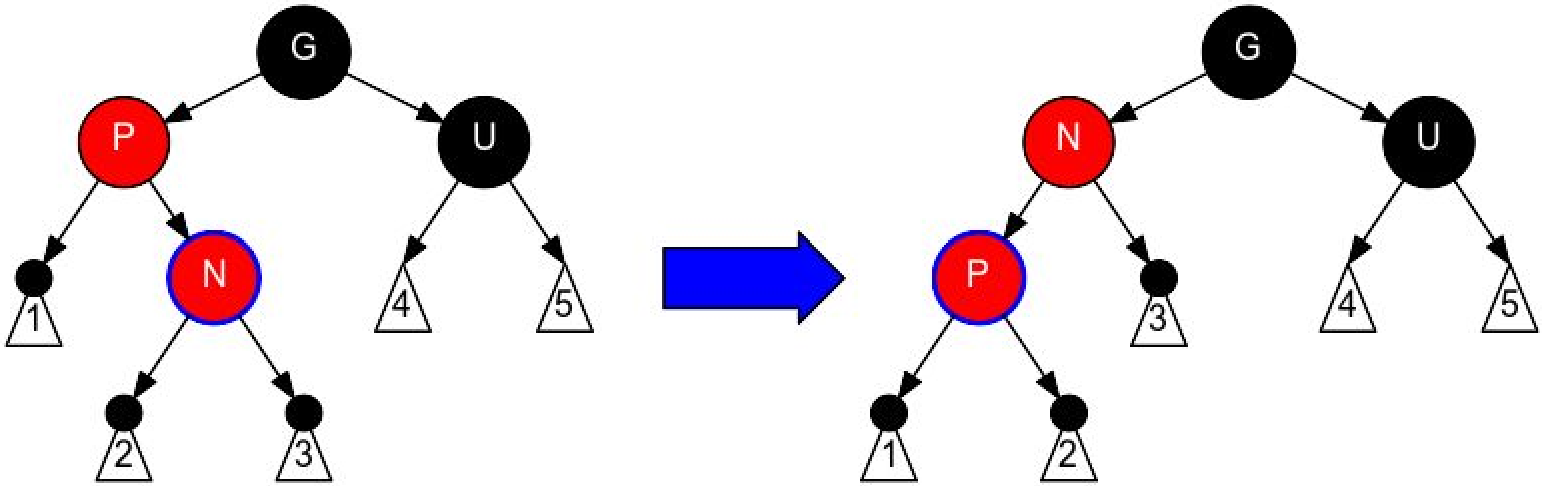
G must have been black, so bg(G) did not change.

If G's parent is red, this may cause additional changes.



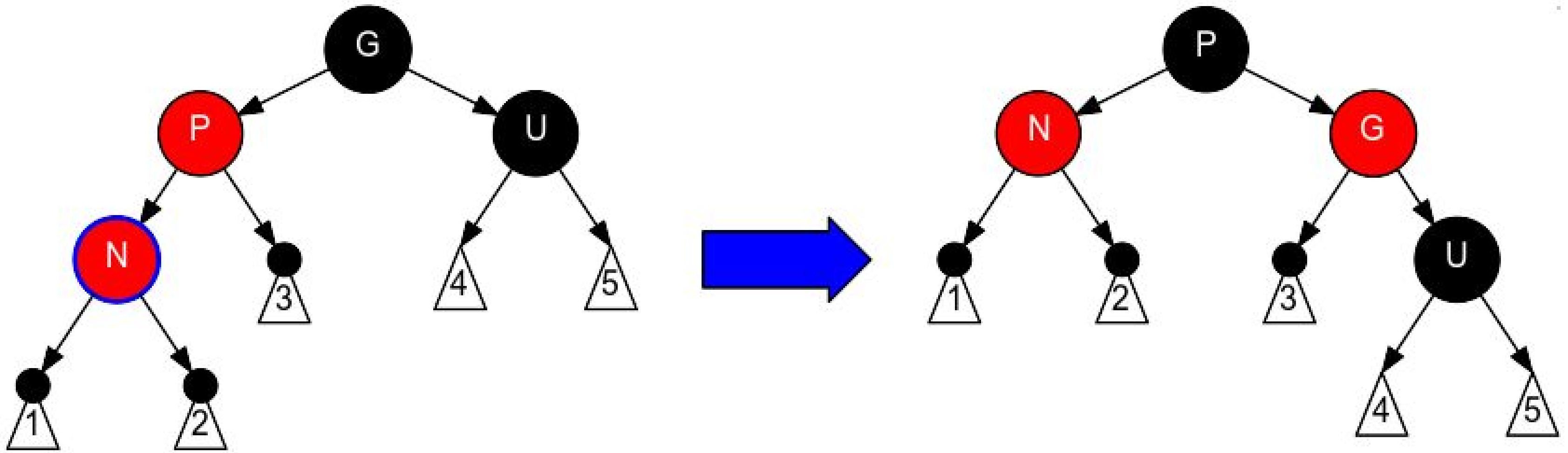
N is Left-Right of G, P is Red, U is Black

Left rotation on P. Converts case 4 to case 5.



N is Left-Left of G, P is Red, U is Black

Right rotation on G, switch colors of P and G.



Deletion

Swap with smallest element from right subtree, do not change color of node, delete the swapped node location.

The smallest element from the right subtree:

1. Satisfies the search tree criteria at the new location
2. Satisfied the red-black properties before the node was deleted

Now we are deleting a node with at most one child.

Note, if there were no right children, use the greatest child of the left subtree and swap all left/right in what follows.

More Labels

M is the node being deleted

C is the child (NULL is fine if there were no proper children)

Cases (we will number the complex case)

M is red - Remove, done.

M is black, C is red - Replace M with C, Recolor C, Done

M is black, C is black (NULL is black) - All the complexity

- Replace M with C, relabel C$\rightarrow$N in the new position
- S is the sibling of N (was the sibling of M)
- S_L is Sibling's left child
- S_R is Sibling's right child

Enumerate

1. N is the new root
2. S is red
3. P, S, and S's children are black
4. S and S's children are black, P is red
5. S is black, S_L is red, S_R is black, N is the left child of P
6. S is black, S_R is red, N is the left child of P

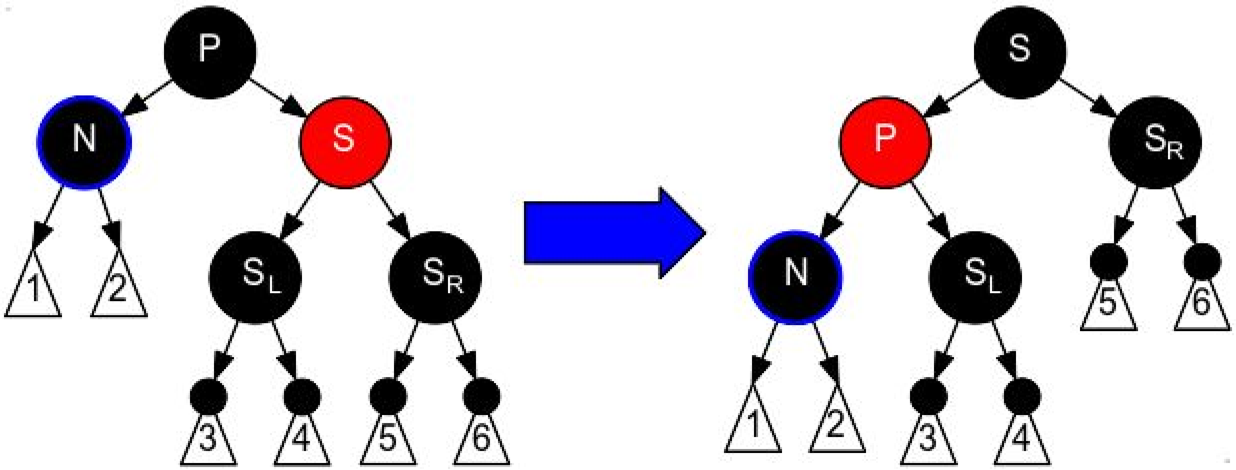
N is the New Root

Done

S is Red

Switch colors of P and S, rotate left at P.

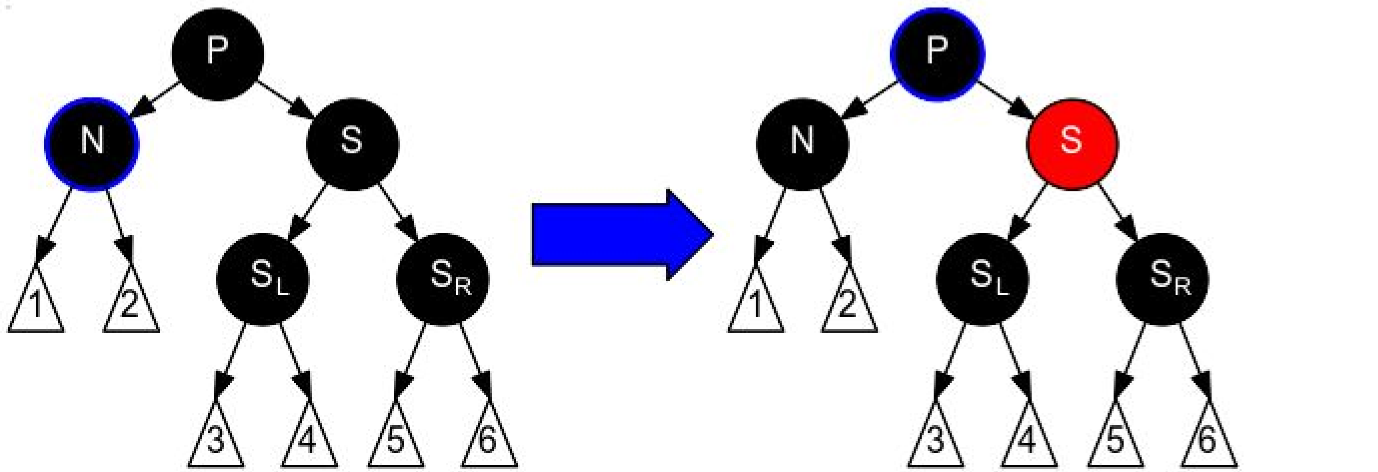
Doesn't fix (missing black node), just transformed to cases 4-6



P, S, and S's Children are Black

Make S red, recurse on P (back to case 1)

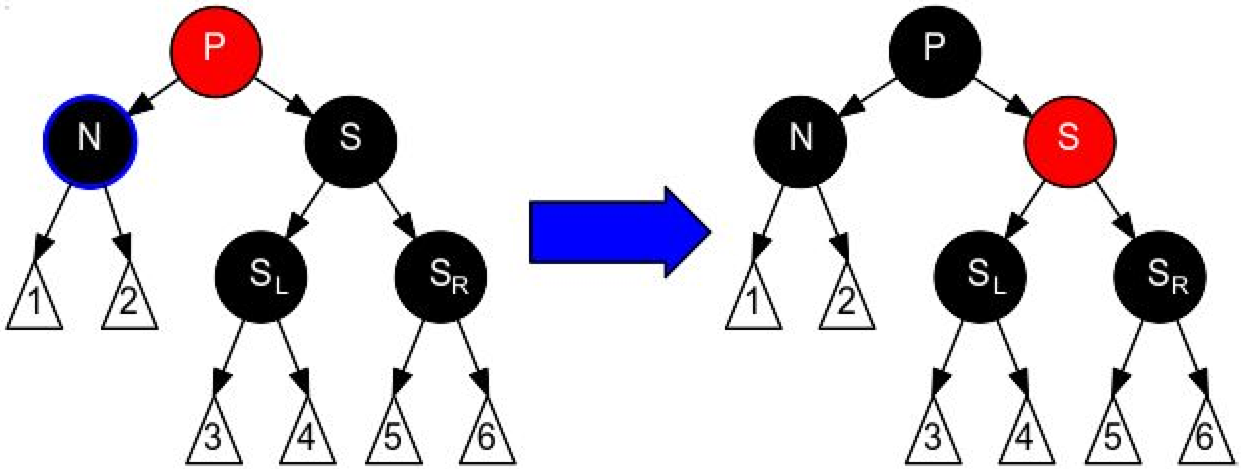
N's side was one black short. P's subtree is fixed, but property 5 is broken unless P was the root.



S and S’s Children are Black, P is Red

Switch colors of P and S, Done

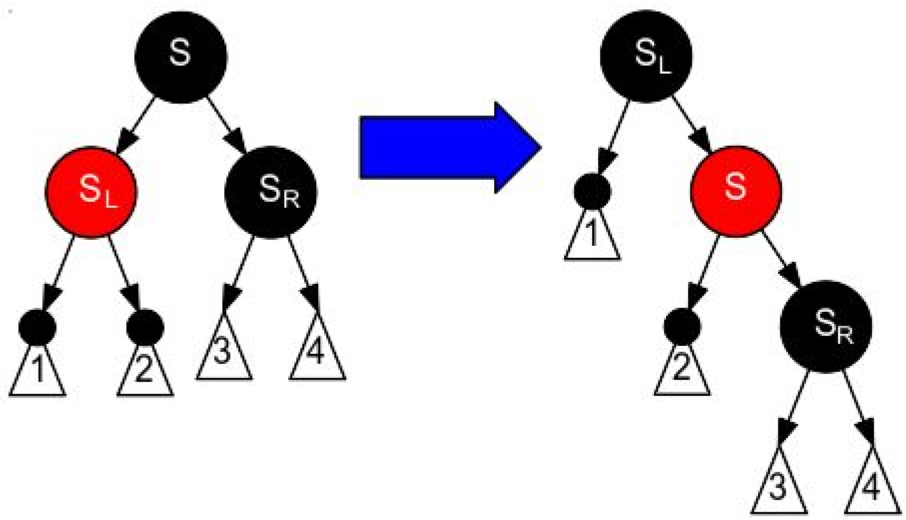
Black-depth of N increased without impacting S’s subtree



S is Black, S_L is Red, S_R is Black, N is left child of P

Rotate right at S, switch colors of S_L and S

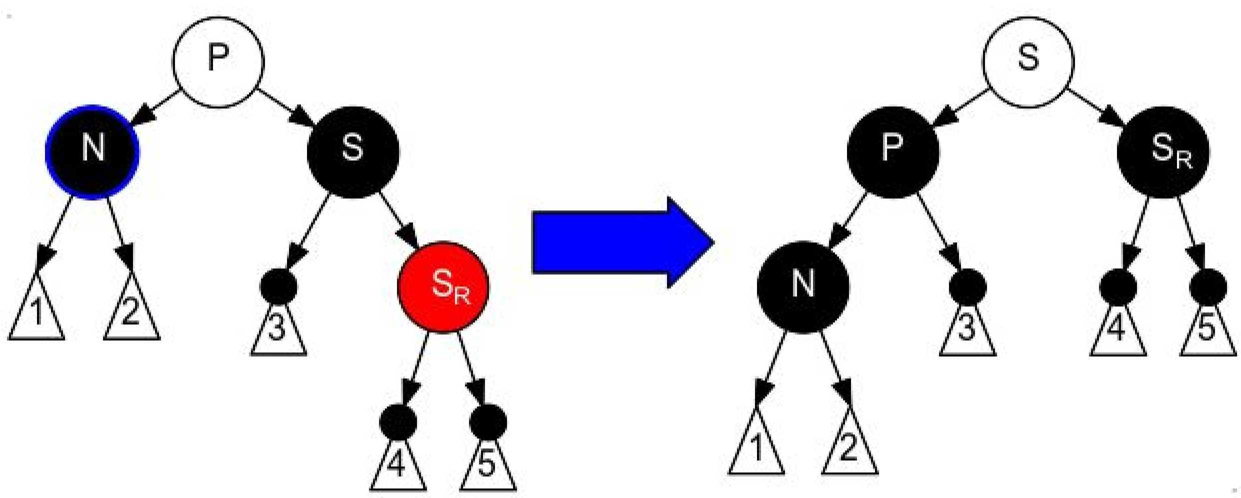
This transforms it into a special form of case 6



S is Black, S_R is Red, N is the left child of P

Rotate left at P, switch colors of S and P, make S_R black

N's black-depth incremented by one, other paths unchanged



Recursion

Wikipedia has tail-recursive code for all of this.

Only specific cases recursed

The number of recursions in theory for a set of insertions/deletions is slightly better than for AVL trees (constant amortized update costs)

In practice, these stay quite balanced