Red-Back Trees

Red-Black trees are an alternative to AVL trees. TreeSet and TreeMap in the Java Collections are implemented using red-black trees.

http://webpages.ull.es/users/jriera/Docencia/AVL/AVL%20tree%20applet.htm

The height of a red-black tree is at most 2 lg(N+1).

Theorem: The height h of a red-black tree with n internal nodes is no greater than $2\log(n+1)$.

Proof:

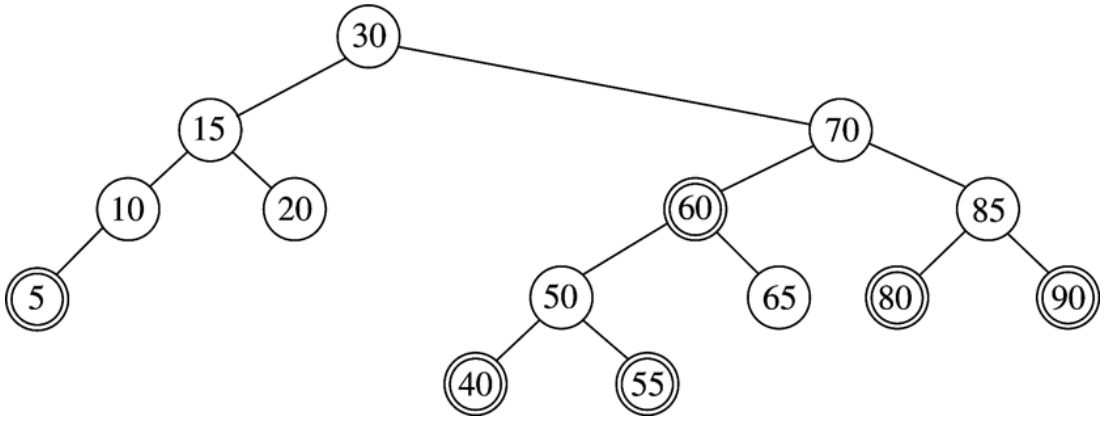
- By property 5, every root-to-leaf path in the tree has the same number of black nodes; let this number be B.
- So there are no leaves in this tree at depth less than B, which means the tree has at least as many internal nodes as a complete binary tree of height B.
- Therefore, $n \geq 2^B - 1$. This implies $B \leq \log(n+1)$.
- By property 4, at most every other node on a root-to-leaf path is red. Therefore, $h \leq 2B$.
- Putting these together, we have $h \leq 2\log(n+1)$.

Q.E.D.

Thus, operations on Red-Black trees are O(lgN) worst case.

A red-black tree is a binary search tree (BST) that satisfies the following conditions:

1. Every node is either red or black.
2. The root is black.
3. If a node is red, then both its children are black. (Or in other words, two red nodes may not be adjacent.)
4. For each node, all paths from the node to a null reference contain the same number of black nodes.



Red nodes are double circles.

Bottom-up insertion

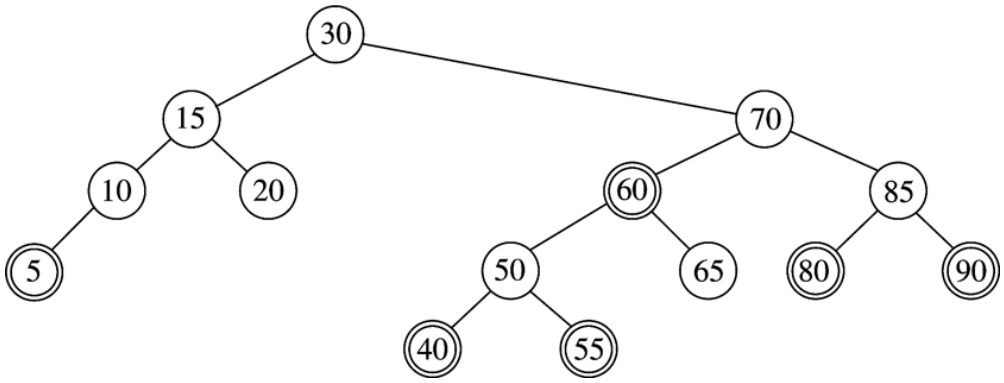
Place new item as leaf, as usual.

If we color item black, we will violate condition 4, because we will create a longer path of black nodes. Thus, the item must be colored red.

If the new item is the root, change its color to black.

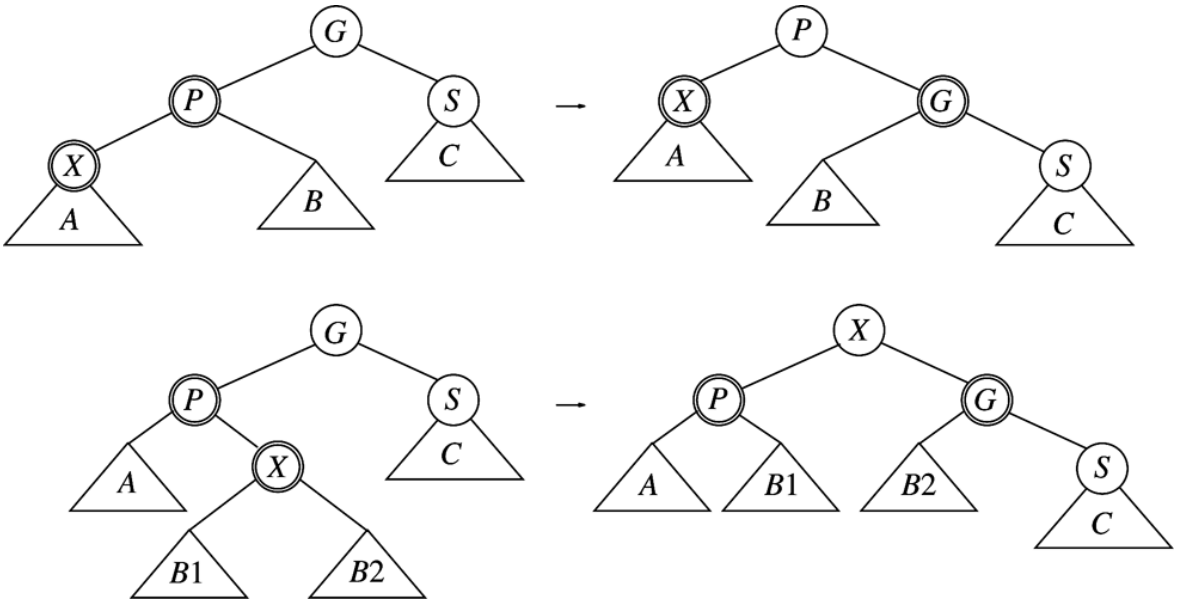
If the parent is black, we are done.

If the parent is red, then we will violate condition 3 by having consecutive red nodes $\Rightarrow$ color changes and rotations.



Insert 25: node 25 is a red right-child of 20; 20 is black; done.

Insert 3: node 3 is a red left-child of 5; 5 is red, so rotate …



X = newly added leaf : red

P = parent of X: red

S = sibling (if it exisits) of parent

G = grandparent of X : must be black since P is red; otherwise there wewre consecutive reds (G & P) before insert.

Rotations are zig-zig or zig-zag.

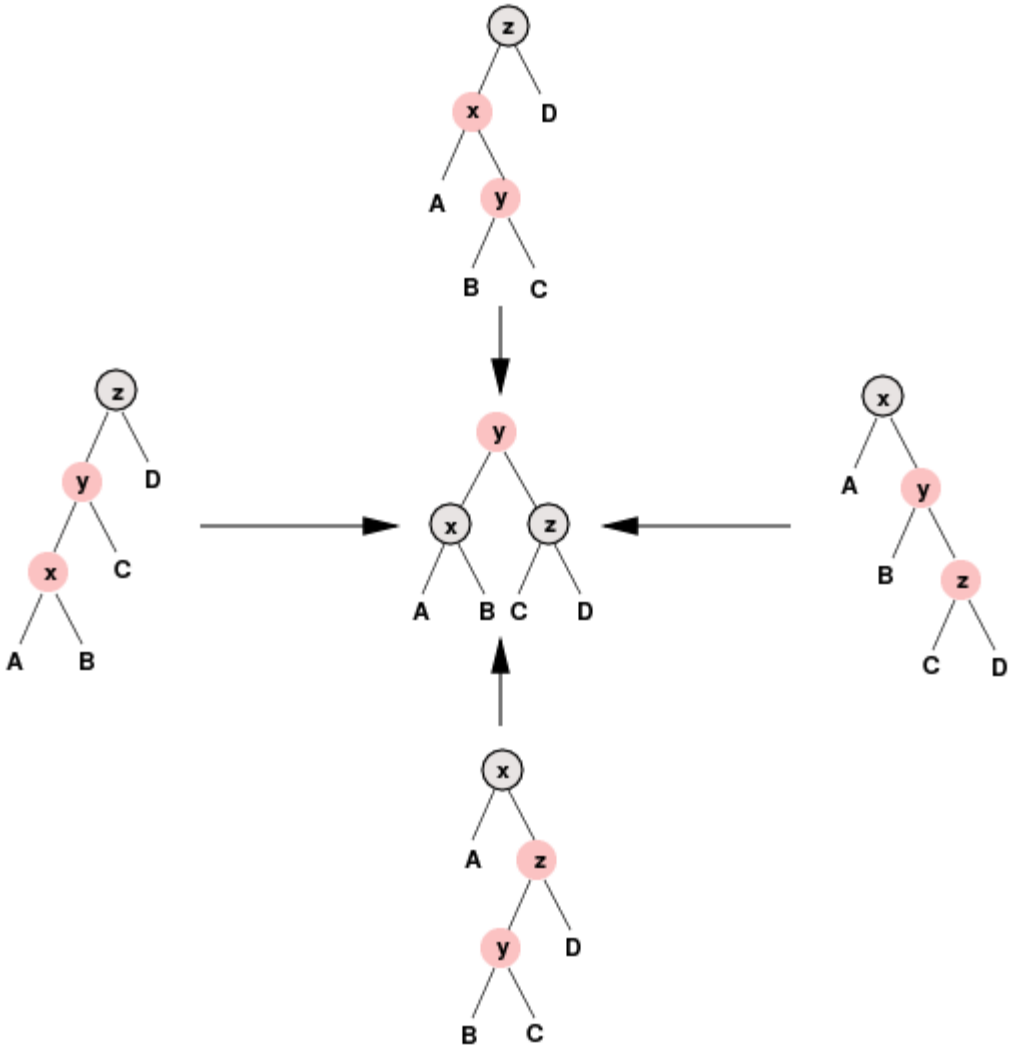
- Zig-zig: single rotation between P and G.
- Zig-zag: double rotation: first between X and P, then between X and G.

$\Rightarrow$ Subtree's new root is black.

So even if great-grandparent was red, not possible to have 2 consecutive reds

Number of black nodes on paths into A, B, and C is unchanged.

Another way to look at it:



Notice that the ordering <*AxByCzD*> is preserved by these transformations.

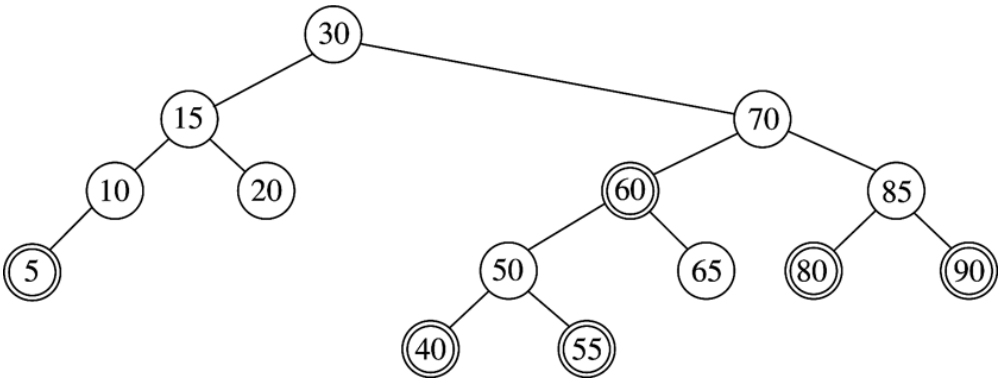
Notice that these transformations do not change the number of black nodes on any path from the parent of this subtree (if one exists) to the leaves.

The only conditions that can be violated are condition 2 (if *y* is the root) and conditon 3 (if *y*'s parent is red).

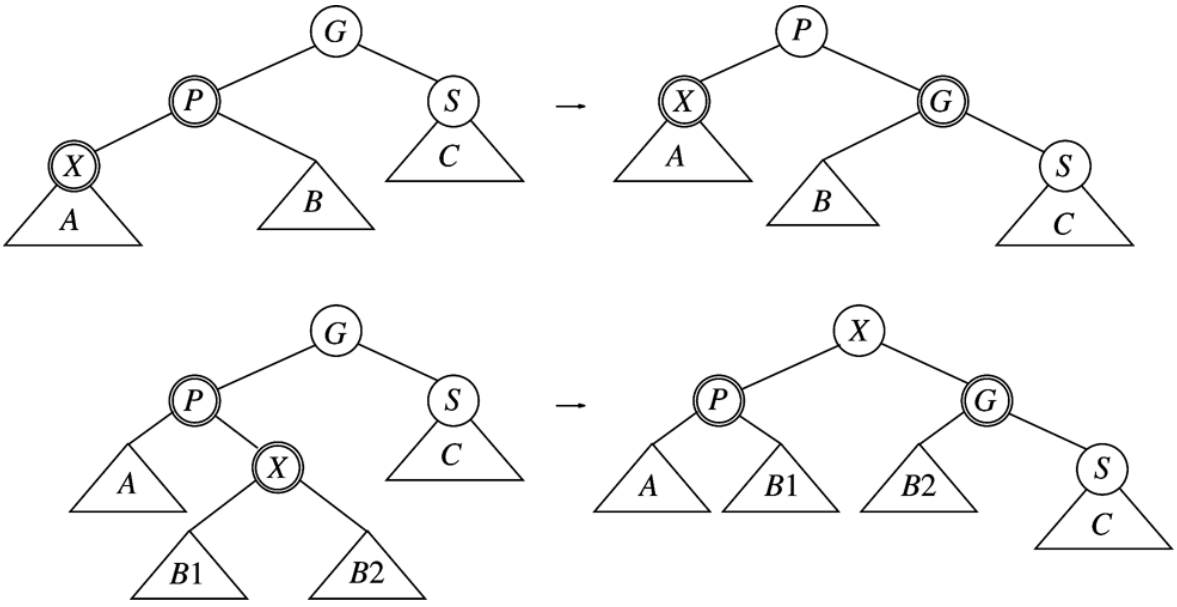
⇒ We can repeat this operation until either *y*'s parent is black, in which case we're done, or *y* is the root, in which case we color *y* black and we're done.

(Coloring the root black increases the number of black nodes on every path from the root to the leaves by the same amount, so property 5 will not be violated after that re-coloring if it wasn't violated before.)

Back to original figure:



Insert 79.

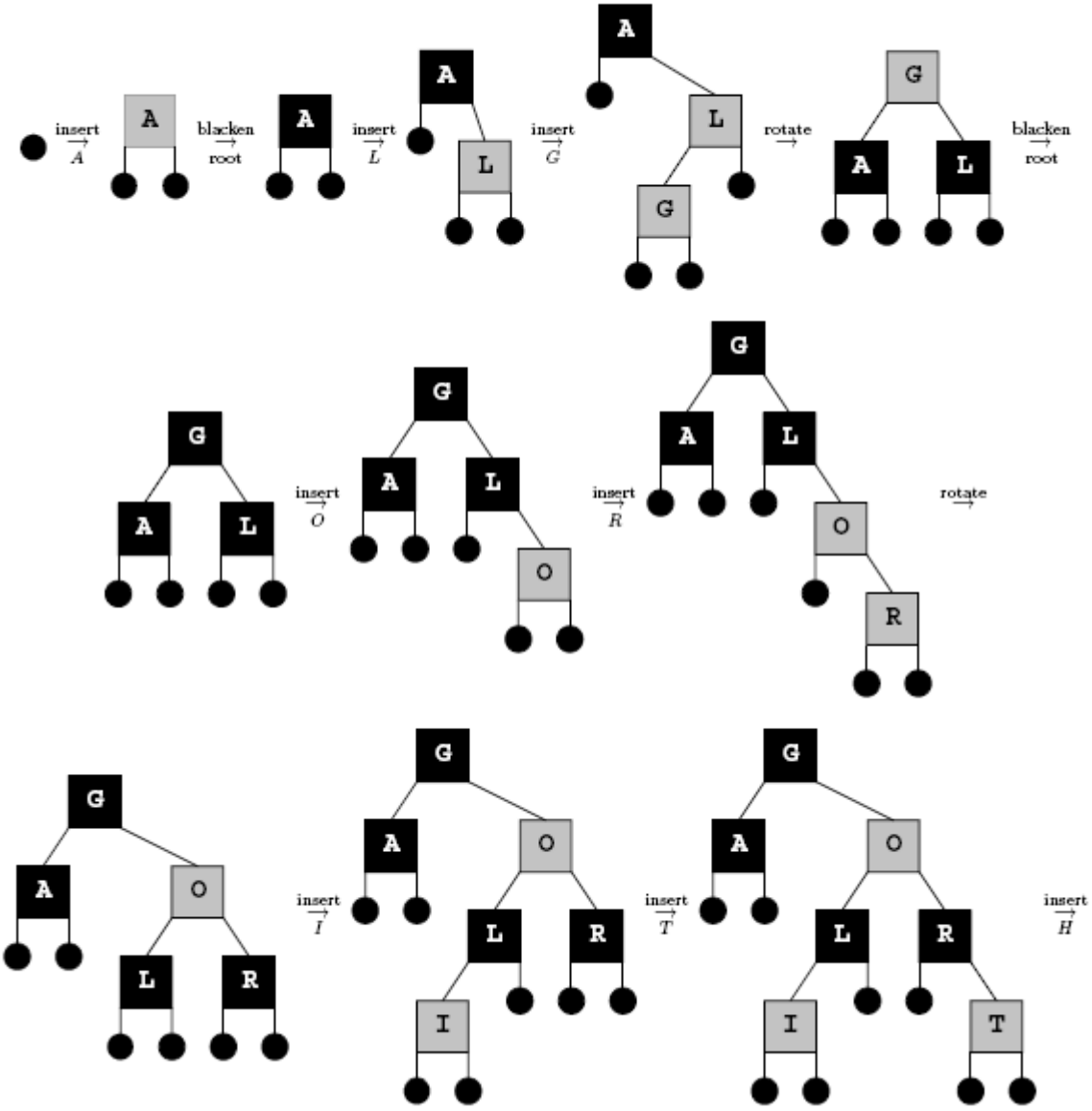


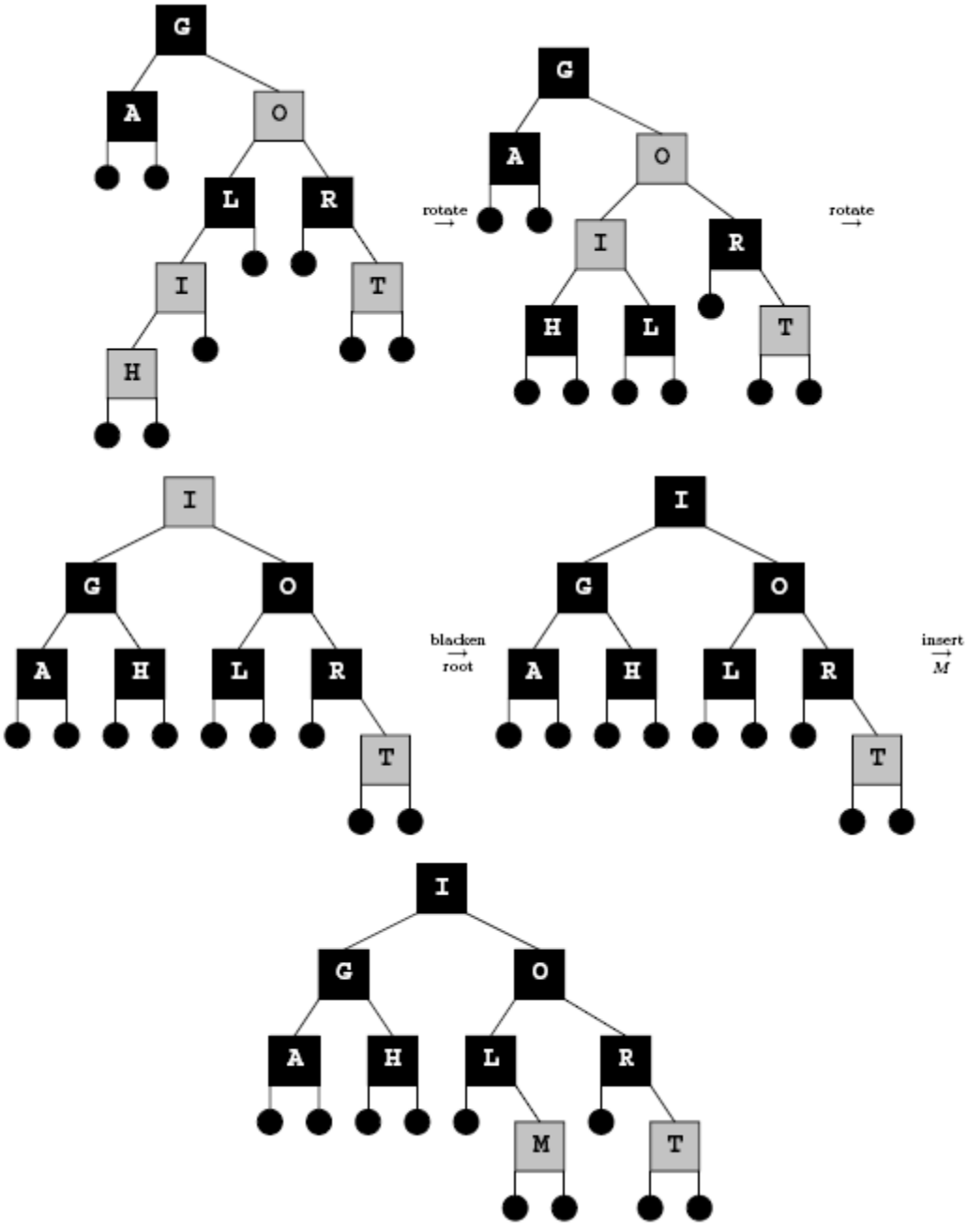
Use Zig-Zig, but S is red, not black:

- initially, there is one black node on the path from the subtree's root to C.
- After rotation, there must still only be only black node, but there are three nodes (new root, G, and S) on the path to C.
- Since only one may be black, and since we cannot have consecutive red nodes, color both S and the new subtree's root red, and G (black)
- But what if great-grandparent is also red ⇒percolate procedure up to root.

Red-Black Tree Insertion Example

Insert the letters `A L G O R I T H M` in order into a red-black tree.




G
A
O
L
R
I
T
H
rotate
G
A
O
I
R
H
L
T
rotate
I
G
O
A
H
L
R
T
blacken
root
I
G
O
A
H
L
R
T
insert
M
I
G
O
A
H
L
R
M
T