

Red-Black Trees and AA Trees

Binary Tree Representation Of 2-3-4 Trees

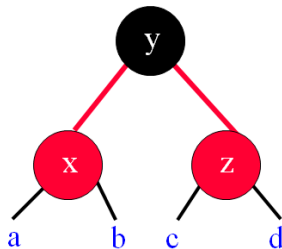
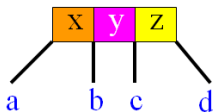
- Problems with 2-3-4 trees.



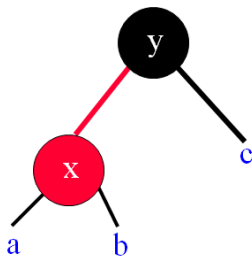
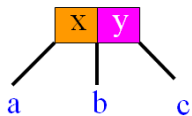
2-3-4 node structure

- 2- and 3-nodes waste space.
- Overhead of moving pairs and pointers when changing among 2-, 3-, and 4-node use.
- Represented as a binary tree for improved space and time performance.

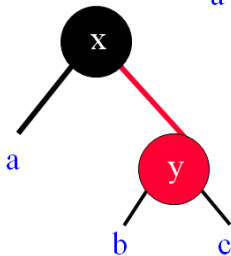
Representation of a 4-node



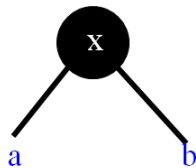
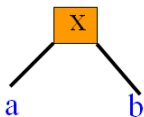
Representation of a 3-node



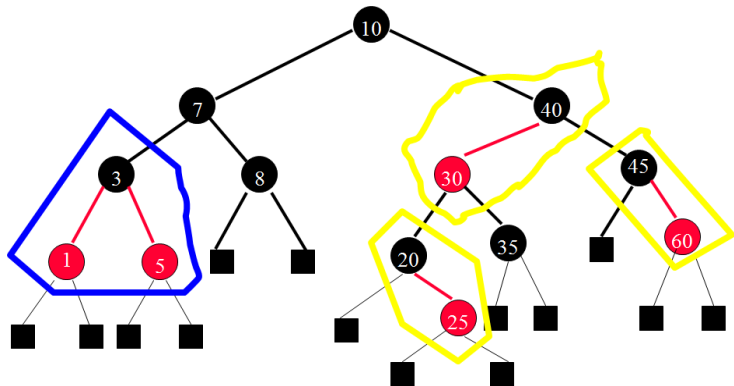
OR



Representation of a 2-node



An Example



Red Black Trees

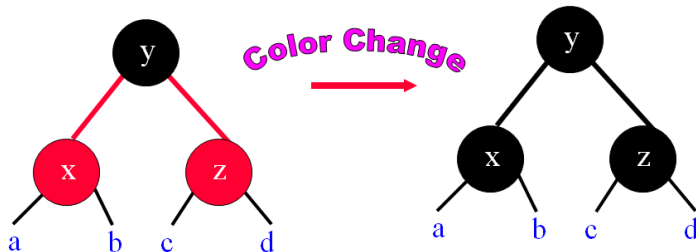
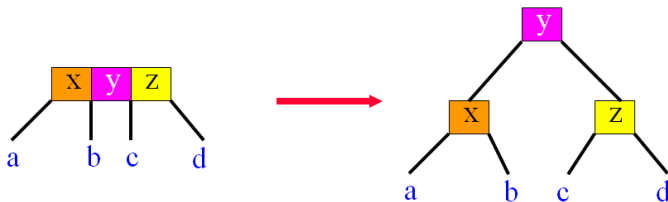
Colored Nodes Definition

- Binary search tree.
- Each node is colored **red** or **black**.
- Root and all external nodes are black.
- No root-to-external-node path has two consecutive **red** nodes.
- All root-to-external-node paths have the same number of **black** nodes
- The height of a red black tree that has n (internal) nodes is between $\log_2(n + 1)$ and $2\log_2(n + 1)$.
- C++ STL implementation
- `java.util.TreeMap` => red black tree

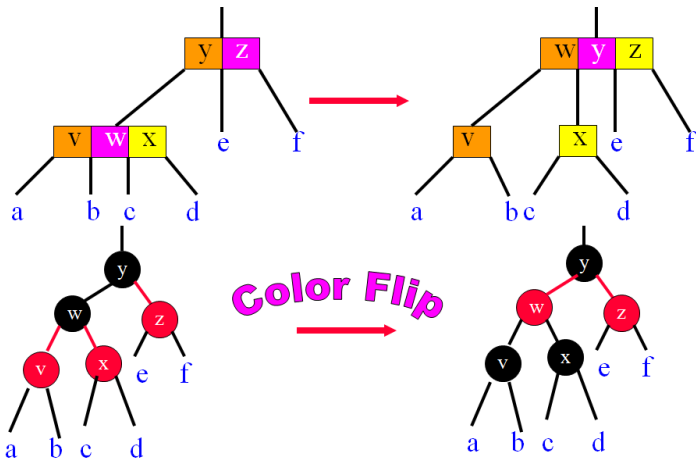
Top-Down Insert

- Mimic 2-3-4 top-down algorithm.
- Split 4-nodes on the way down.

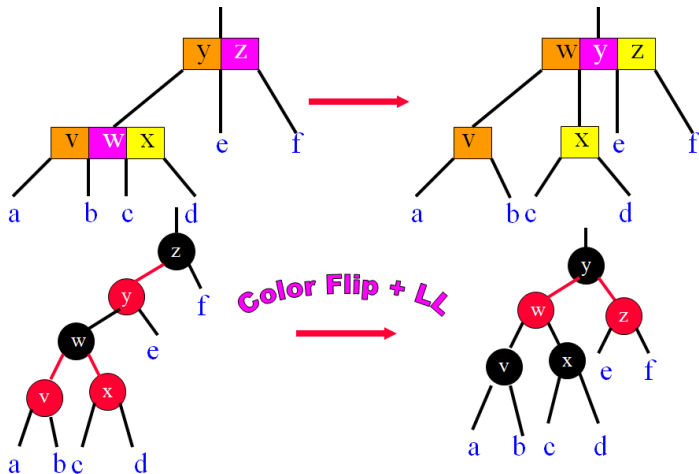
Root Is a 4-node



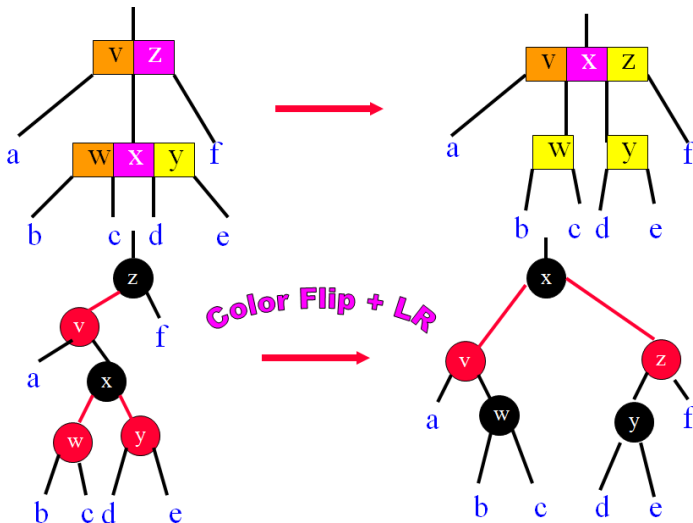
4-node Left Child of 3-node



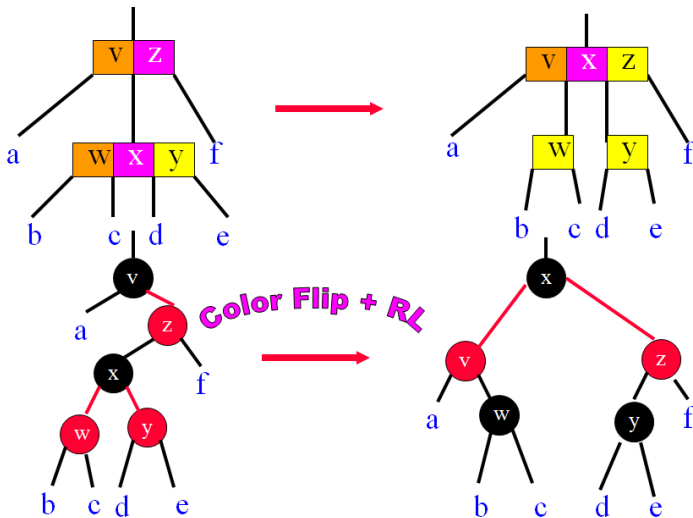
4-node Left Child of 3-node



4-node Middle Child of 3-node



4-node Middle Child of 3-node

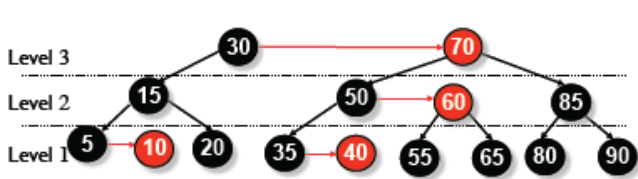


4-node Right Child Of 3-node

- One orientation of 3-node requires color flip.
- Other orientation requires RR rotation.

AA Trees

- An *AA tree* satisfies the properties of Red-Black trees plus one more:
 - ▶ Every node is colored either red or black
 - ▶ The root is black
 - ▶ If a node is red, both of its children are black.
 - ▶ Every path from a node to a null reference has the same number of black nodes
 - ▶ Left children may NOT be red
- Invented by A. Andersson in 1993.



Advantage of AA Trees

- AA trees simplify the algorithms
 - ▶ It eliminates half the restructuring cases
 - ▶ It simplifies deletion by removing an annoying case
 - ★ if an internal node has only one child, that child must be a red right child
 - ★ We can always replace a node with the smallest child in the right subtree (it will either be a leaf or have a red child)

Representing the Balance information

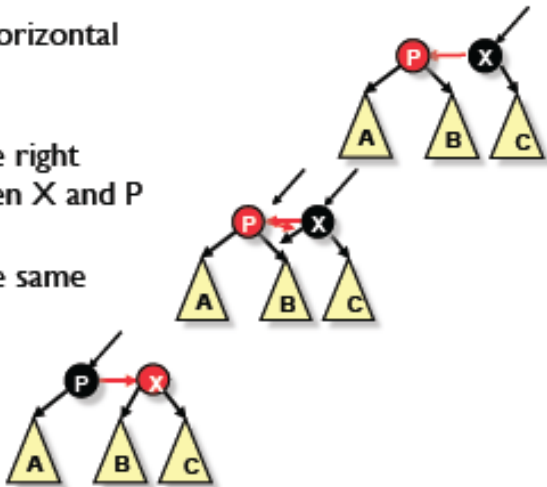
- In each node we store a *level*. The level is defined by these rules
 - ▶ If a node is a leaf, its level is 1
 - ▶ If a node is red, its level is the level of its parent
 - ▶ If a node is black, its level is one less than the level of its parent
- The *level* is the number of left links to a null reference.
- A horizontal link is a connection between a node and a child with equal levels
 - ▶ Horizontal links are right references
 - ▶ There cannot be two consecutive horizontal links
 - ▶ Nodes at level 2 or higher must have two children
 - ▶ If a node has no right horizontal link, its two children are at the same level

skew - remove left horizontal links

Problem: left horizontal link in AA-tree

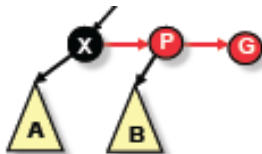
Skew is a simple right rotation between X and P

P remains at the same level as X

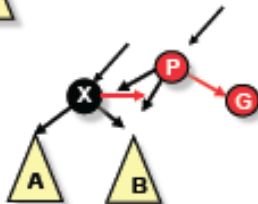


split - remove consecutive horizontal links

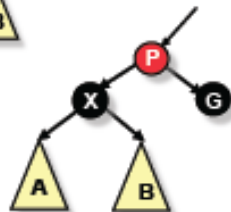
Problem: With G inserted, there are two reds in a row



Split is a simple left rotation between X and P



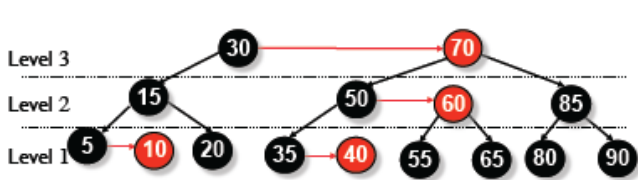
P's level increases in the AA-tree



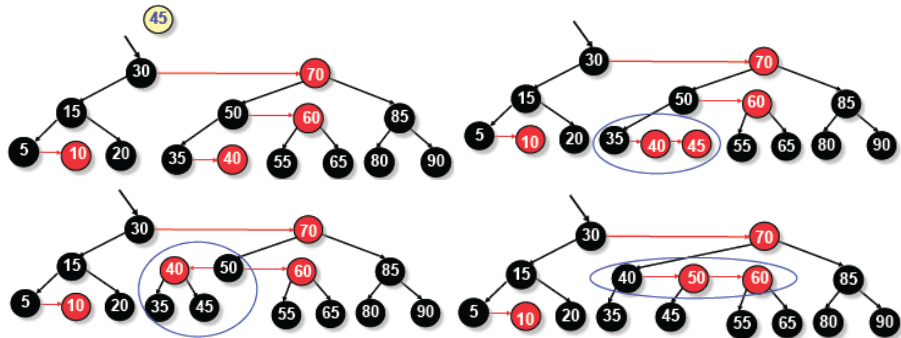
- A *skew* removes a left horizontal link
- A *skew* might create consecutive right horizontal links
- We should first process a *skew* and then a *split*, if necessary
- After a *split*, the middle node increases a level, which may create a problem for the original parent

Insertion

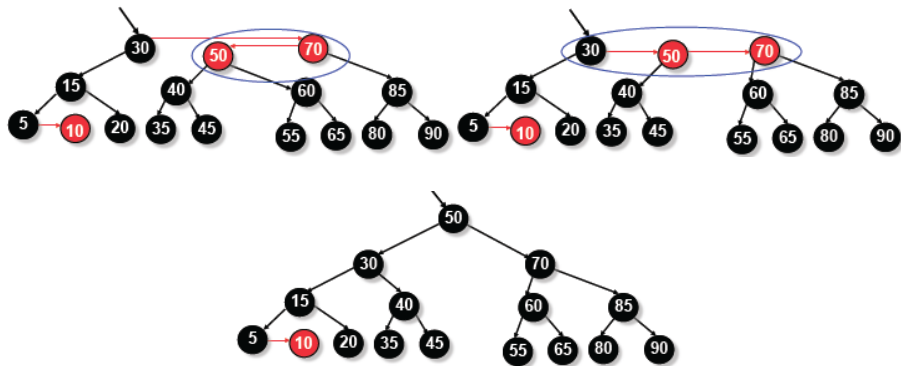
- A new item is always inserted at the bottom level
- In the previous example, inserting 2 will create a horizontal left link
- In the previous example, inserting 45 generates consecutive right links
- After inserting at the bottom level, we may need to perform rotations to restore the horizontal link properties



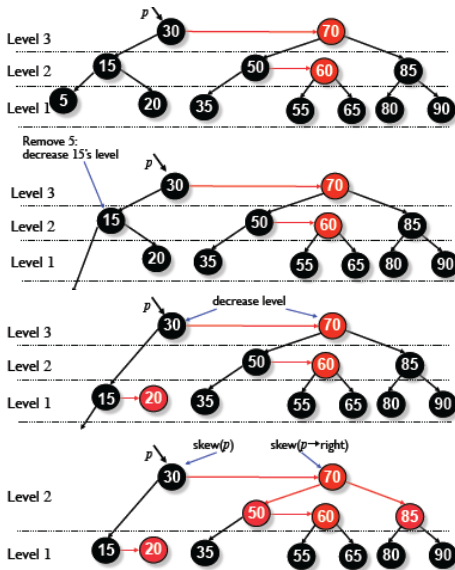
An Example - Insert 45



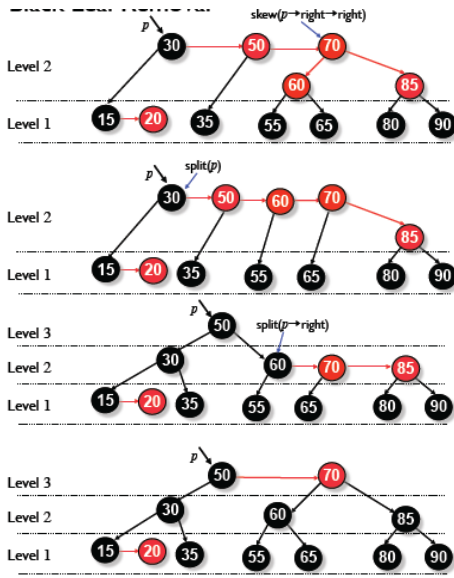
An Example - Insert 45 (Cont'd)



An Example - Delete 5



An Example - Delete 5 (Cont'd)



Summary

AVL Trees: maintain balance factor by rotations

2-3 Trees: maintain perfect trees with variable node sizes using rotations

2-3-4 Trees: simpler operations than 2-3 trees due to pre-splitting and pre-merging nodes, wasteful in memory usage

Red-black Trees: binary representation of 2-3-4 trees, no wasted node space but complicated rules and lots of cases

AA-Trees: simpler operations than red-black trees, binary representation of 2-3 trees