AA Trees

another alternative to AVL trees

Balanced Binary Search Trees

- A Binary Search Tree (BST) of N nodes is balanced if height is in ***O****(log* N)
- A balanced tree supports efficient operations, since most operations only have to traverse one or two root-to-leaf paths.
- There are many implementations of balanced BSTs, including AVL trees, Red-Black trees and AA trees.

Properties of AA Trees

- An ***AA tree*** satisfies the properties of Red-Black trees plus one more:
  1. Every node is colored either red or black
  2. The root is black
  3. If a node is red, both of its children are black.
  4. Every path from a node to a *null* reference has the same number of black nodes
  5. Left children may NOT be red

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]

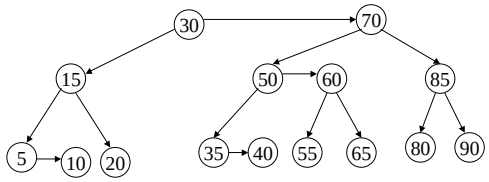
Representing the Balance information

- 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.

Links in an AA tree

- 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

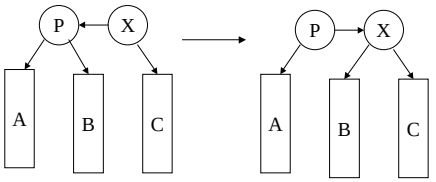
Example of an AA Tree



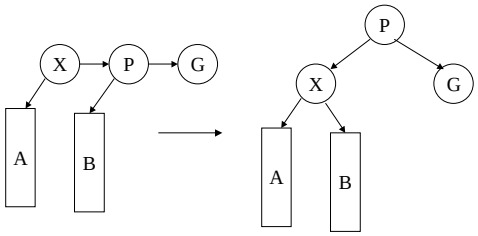
Insertion

- 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

`skew` - remove left horizontal links



`split` - remove consecutive horizontal links



`skew/split`

- 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