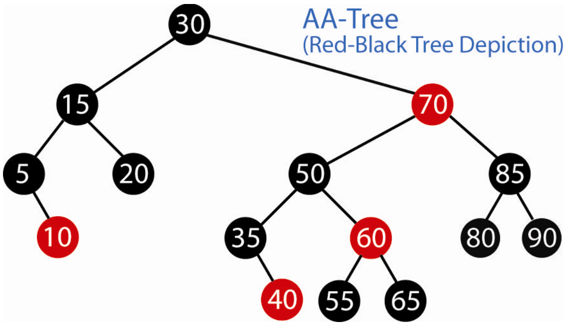
## 19.6 – AA-Trees

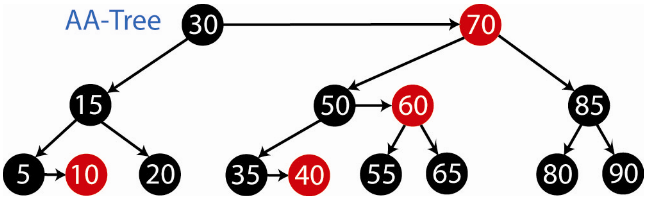
1. Interactive demo of a number of tree structures: http://people.ksp.sk/~kuko/bak/

2. An ***AA-tree*** is a red-black tree such that no left child is red (*i.e* red nodes must be right children). This restriction greatly simplifies the insert and remove algorithms.



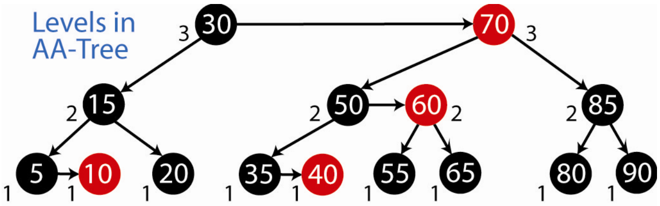
3. The implementation of this idea is simplified by re-introducing balance information in the following way:

   a. Red children are at the same level as their parent
   b. Black children are below their parent



4. The *level* of a node is the number of left links on the path to a null node. An AA-tree can then be defined as follows. The level of a node is:

   a. 1, if the node is a leaf
   b. the level of its parent, if node is red
   c. one less than the level of its parent, if node is black



   In an AA-tree, with the addition of the level information, we no longer need the red/black coloring information, so it is eliminated from the coming algorithms. We will retain it in some of the pictures for emphasis.

5. Implications of definition

    a. Left children must be one level lower than parent. (Left can't be red).
    b. Right children can be at same level as parent (if red), or below (if black)

    A *horizontal link* indicates two nodes at the same level. Horizontal links must be right links (only right children can be red, and red's are at the same level as parent)

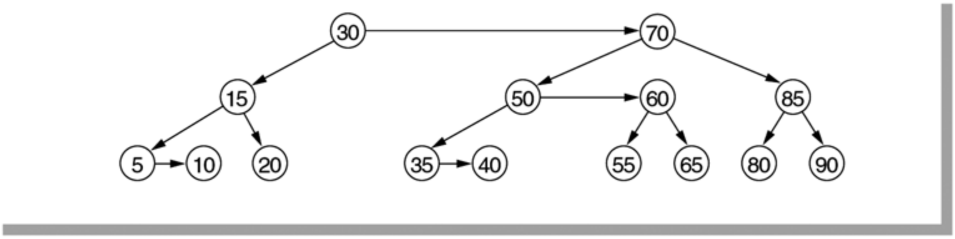
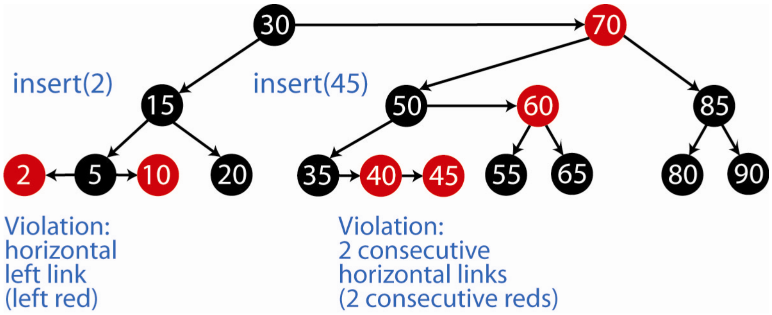


**figure 19.54**

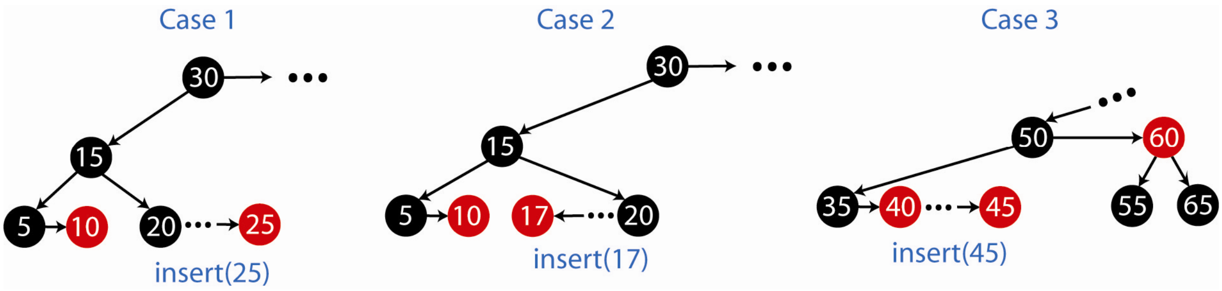
AA-tree resulting from the insertion of 10, 85, 15, 70, 20, 60, 30, 50, 65, 80, 90, 40, 5, 55, and 35

    c. There cannot be two consecutive horizontal links (can't have 2 be two consecutive red nodes)
    d. Nodes at level 2 or higher must have 2 children
    e. If a node does not have a right horizontal link, then its two children are on the same level.

6. As with the Red-Black tree, we always insert a red node. This can lead to two types of problems. Consider inserting 2 or 45 into the tree above.



Thus, when we insert a node, there are three cases:



a. Case 1 – If the node to insert is to the right of its parent, then we insert it at the same level, as a horizontal (right) link. If the grandparent is at a higher level, then we are done.

b. Case 2 – If the node to insert is to the left of its parent, then it will be at the same level as its parent, which is a violation. To fix a horizontal left link, we use a procedure called *skew*.

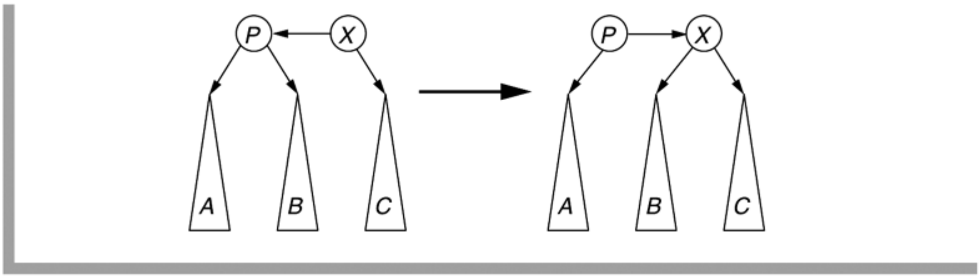
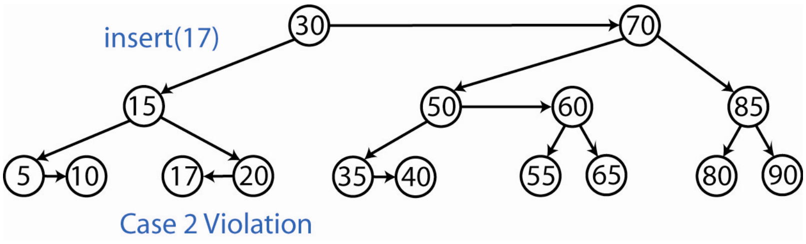
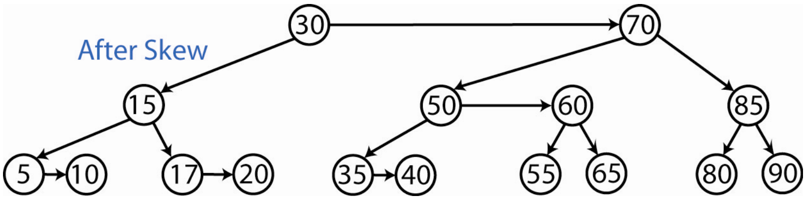


**figure 19.55**

The `skew` procedure is a simple rotation between $X$ and $P$.

Example:

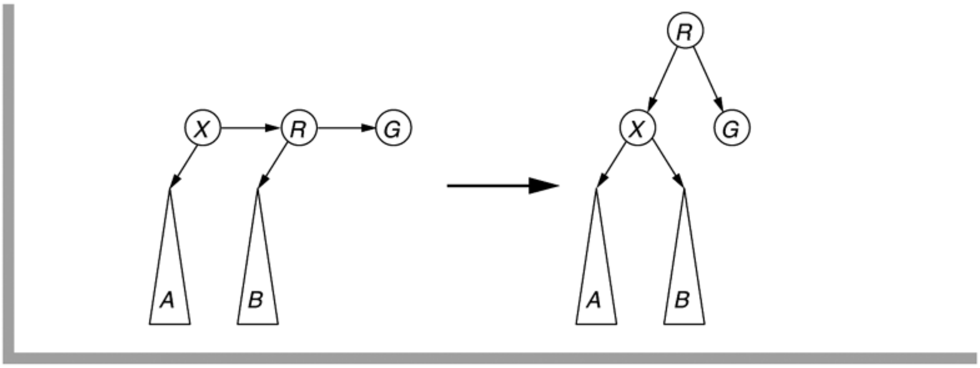




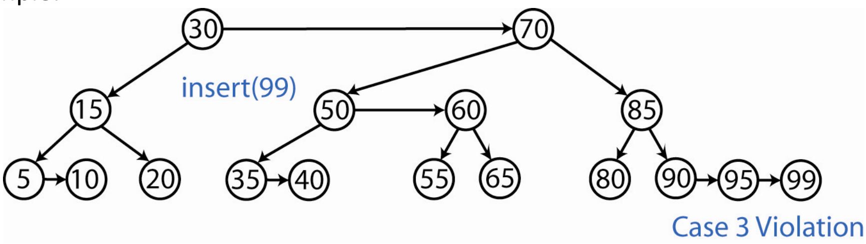
c. Case 3 – If the grandparent is at the same level, then we have 2 consecutive horizontal links (reds), which is a violation. This is fixed by a procedure called *split.*

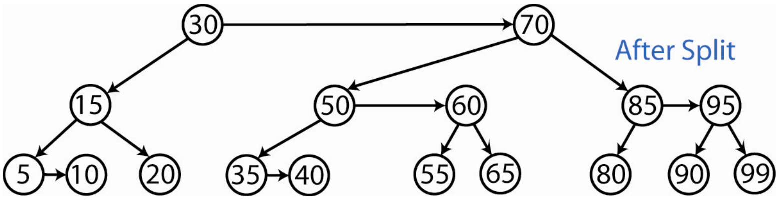
**figure 19.56**

The `split` procedure is a simple rotation between $X$ and $R$; note that $R$'s level increases.



Example:





7. Example: Sometimes a Split or Skew will introduce a new violation. So, we continue to Split or Skew until there are no violations.

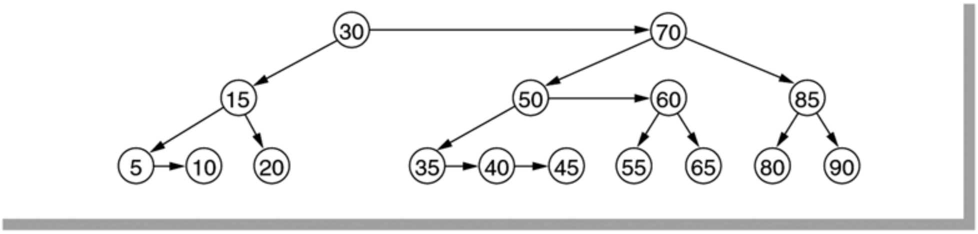


**figure 19.57**

After insertion of 45 in the sample tree; consecutive horizontal links are introduced, starting at 35.

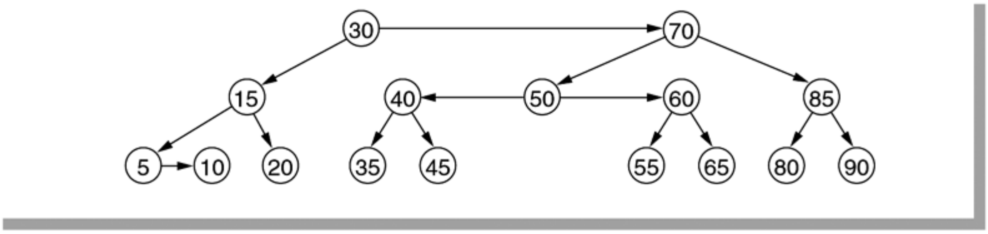


**figure 19.58**

After `split` at 35; a left horizontal link at 50 is introduced.

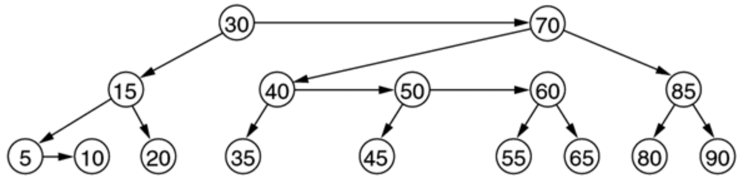


**figure 19.59**

After `skew` at 50; consecutive horizontal nodes are introduced starting at 40.

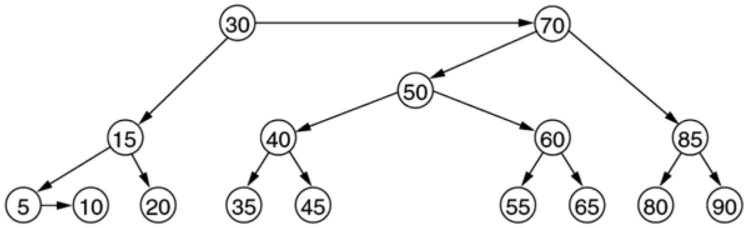


**figure 19.60**

After `split` at 40; 50 is now on the same level as 70, inducing an illegal left horizontal link.

**figure 19.61**

After `skew` at 70; consecutive horizontal links are introduced, starting at 30.

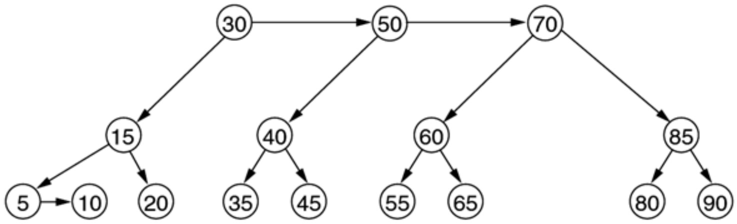
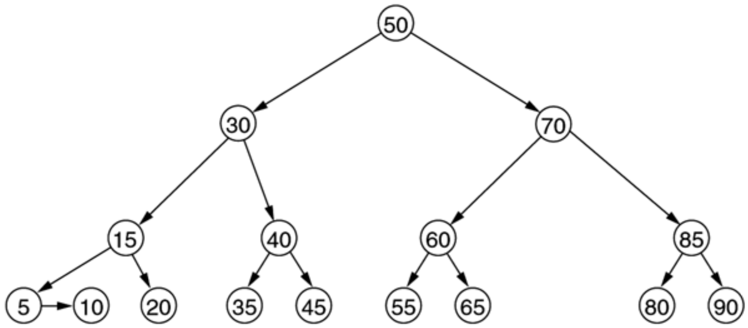


**figure 19.62**

After `split` at 30; the insertion is complete.



8. Below is an algorithm for insert for AA-tree. This is not the way we would implement this, but it shows the overall idea.

```
insert( key ) : newNode
{
    return insert( root, key )
}

insert( node, key ) : newNode
{
    while node is not null

        if key is less than node
            if node has left child
                Advance to left child: node = node.left
            else
                X = new Node(key)
                Connect node to new node: node.left = X
                Repeat until no violations:
                    Skew if necessary
                    Split if necessary
                return X

        if key is greater than node
            if node has right child
                Advance to right child: node = node.right
            else
                X = new Node(key)
                Connect node to new node: node.right = X
                Repeat until no violations:
                    Skew if necessary
                    Split if necessary
                return X
```

9. Example: Build this tree:

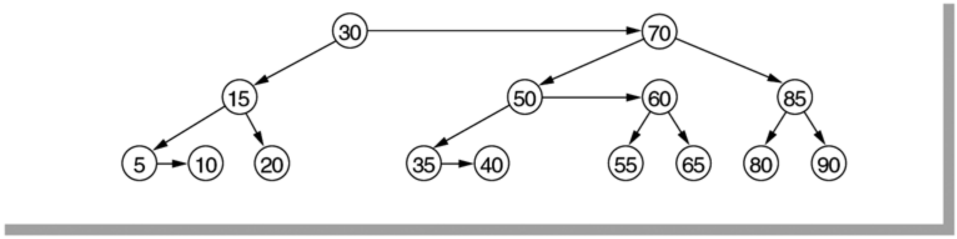


**figure 19.54**

AA-tree resulting from the insertion of 10, 85, 15, 70, 20, 60, 30, 50, 65, 80, 90, 40, 5, 55, and 35

## Homework 19.12

1. Build an AA tree by inserting nodes in this order: 1,2,3,4,5,6,7,8,9