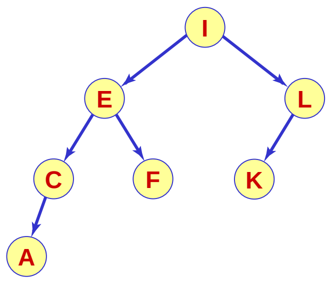
Non-Linear Data Structures

AVL Balanced Trees

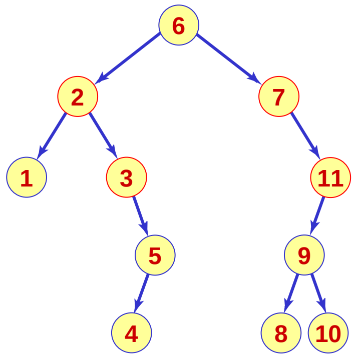
AVL Tree Definition

- An **AVL Tree** is a special case of a height balanced binary tree, where for each node in the tree, the difference in height of its two subtrees is at most one.
- A perfectly balanced binary tree is an AVL tree but not the reverse.
- The operations defined for binary search trees work also for AVL trees, except for the insert and delete operations, where a rebalance may be needed after the operation is done.
- AVL trees have an average search length that is almost identical to the perfectly balanced (minimum height) trees.
- The height of an AVL tree never exceeds 1.45 ($\log_2 n$).

AVL Tree Examples

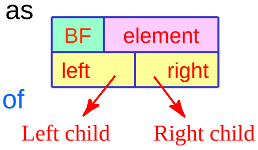


AVL tree

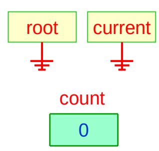


Not an AVL tree

AVL Tree Implementation Using a Linked Structure



- Use the binary tree node generic class: BTNode<E>, as defined before. Add an integer balance factor field.
- The balance factor of a node is defined as the height of its right subtree minus the height of its left subtree.
- Use two BTNode<E> pointer variables:
  - root -- points to the root node of the tree.
  - current -- points to the current element node in the tree.
- Use one integer variable:
  - count -- stores the number of nodes in the tree.
- An empty AVL tree is initialized by setting: root = current = null, and count = 0.



AVL Tree Implementation
Using a Linked Structure (cont.)

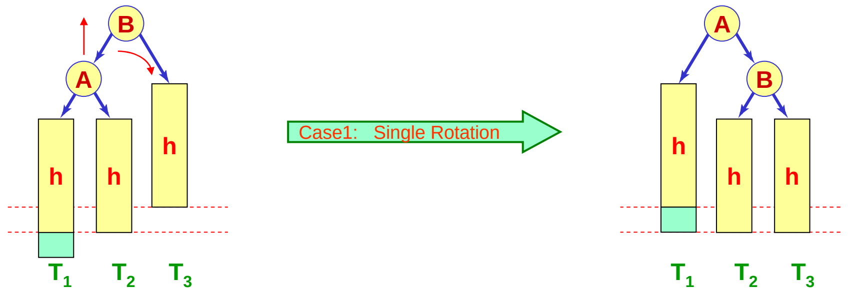
- Operations implemented for the BST are still applicable to the AVL tree.
- The insert, and remove operations need slight changes.
- After the insertion or removal of a node, check for the AVL balance conditions and adjust the tree accordingly.

AVL Tree
The Insert Operation

- Given an AVL tree with a root node, n.
  Let L and R be the left and right subtrees of n, respectively.
  Let h_L and h_R be the heights of L and R, respectively.
- Suppose that a new node is to be inserted in the left subtree, L, then three cases are identified:
  1. $h_L = h_R$: L & R become of unequal height, the tree is still AVL.
  2. $h_L < h_R$: L & R become of equal height, the tree is still AVL.
  3. $h_L > h_R$: The balance is violated, the tree must be rebalanced.
- The rebalancing action is a transformation to restore the balance, where the only movements allowed are in the vertical direction, called rotation. Two cases are possible:

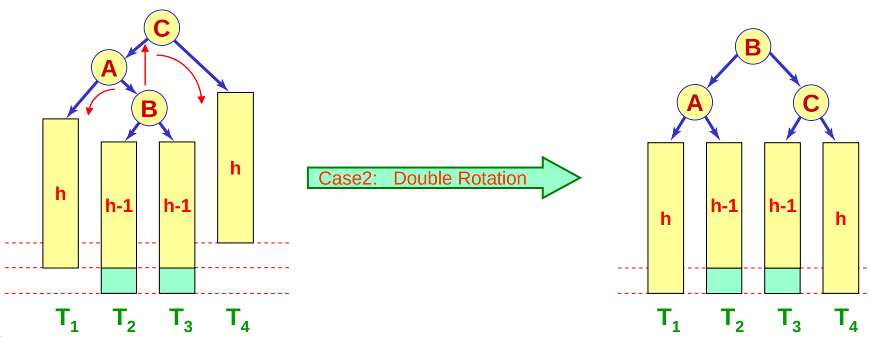
AVL Tree
Case1: The Single Rotation

- The new node is inserted at the leftmost branch of the tree, as shown.
- A similar case exists when the new node is inserted at the rightmost branch of the tree (the mirror image).
- Rebalancing for both cases is done through a single rotation operation.



AVL Tree
Case2: The Double Rotation

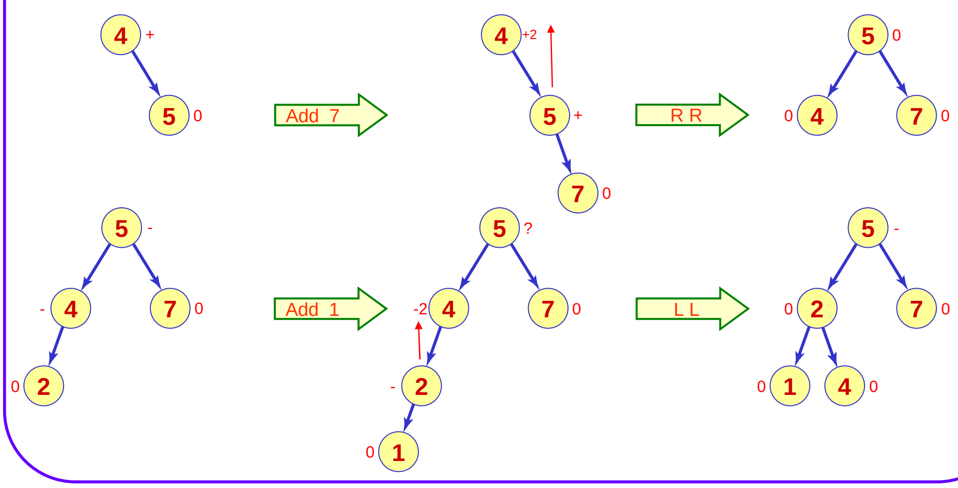
- A new node inserted in the middle of left branch of the tree, as shown.
- A similar case exists when a new node is inserted in the middle of right branch of the tree (the mirror image).
- Rebalancing for both cases is done through a double rotation operation.



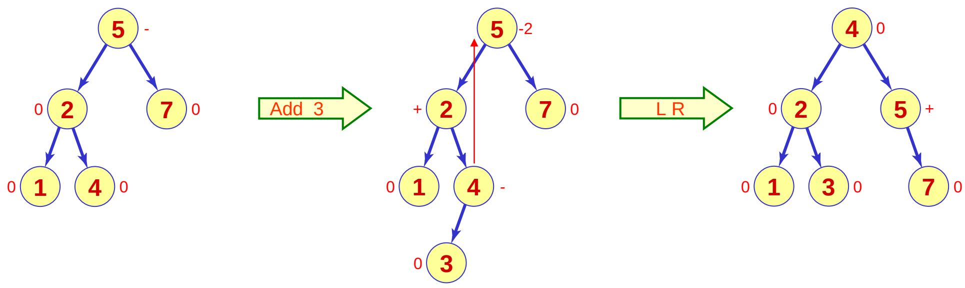
AVL Tree
The Insert Algorithm

1. Follow the search path until it is verified that the key to be inserted is not in the tree.
2. Insert the new node, and determine the resulting balance factor.
3. Retreat along the search path and check the balance factor at each node:
   - Suppose that a node, p, is reached from the left branch with indication that it increased its height (BF = -2).
   - Check the balance factor of the left child of p:
     1. If it is -1, this is case1 ➔ rebalance using a single rotation.
     2. Otherwise, this is case2 ➔ rebalance using double rotation.
   - Similarly, for the right branch, just reverse all the red color text.

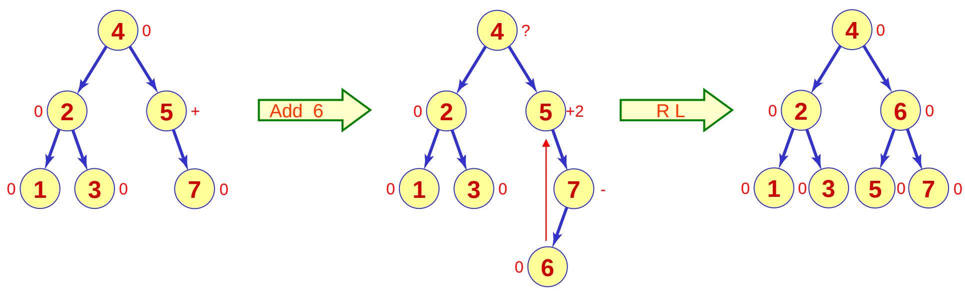
AVL Tree
Examples of the Insert Operation



AVL Tree
Examples of the Insert Operation (cont.)



AVL Tree
Examples of the Insert Operation (cont.)



AVL Tree
The Insert Operation -- Performance

- The expected height of a constructed AVL tree when all n! permutations of n keys occur with equal probability is difficult to get mathematically, but test results show it to have an average:

$$h = \log_2(n) + C, \qquad (C \cong 0.25)$$

- The rebalancing operation is O(1)
- Rebalancing is needed once for approximately every two insertions.
- Use AVL trees only if **searching** is more frequent than insertion.

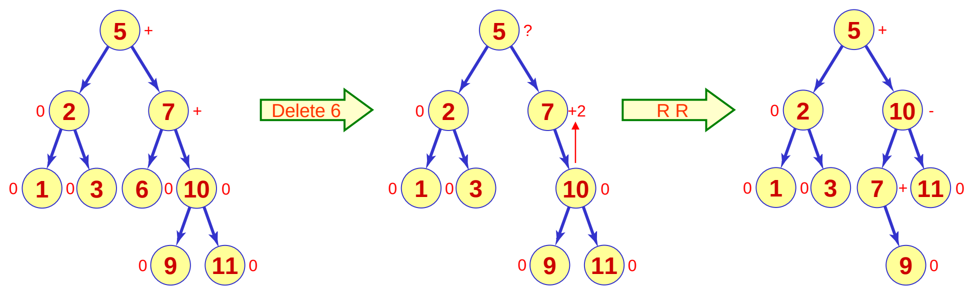
AVL Tree
The Remove Operation

- Given an AVL tree with a root node, n.
  Let L and R be the left and right subtrees of n, respectively.
  Let h_L and h_R be the heights of L and R, respectively.
- Suppose that a node is to be removed from the left subtree, L, then three cases are identified:
  1. $h_L = h_R$: L & R become of unequal height, the tree is still AVL.
  2. $h_L > h_R$: L & R become of equal height, the tree is still AVL.
  3. $h_L < h_R$: The balance is violated, the tree must be rebalanced.
- Rebalancing is done through the same rotation operations used for insertion, namely, single or double rotations.

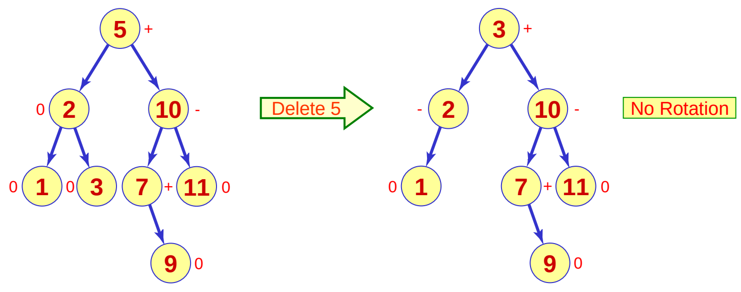
AVL Tree
The Remove Algorithm

1. Follow the search path until it is verified that the key to be removed is in the tree.
2. Remove the node.
3. Retreat along the search path and check the balance factor at each node:
   - Suppose that a node, p, is reached from the left branch with indication that it changed its height (BF = ± 2).
   - Check the balance factor of the right child of p:
     1. If it is >=0, this is case1 ➔ rebalance using a single rotation.
     2. Otherwise, this is case2 ➔ rebalance using double rotation.
   - Similarly, for the right branch, just reverse all the red color text.

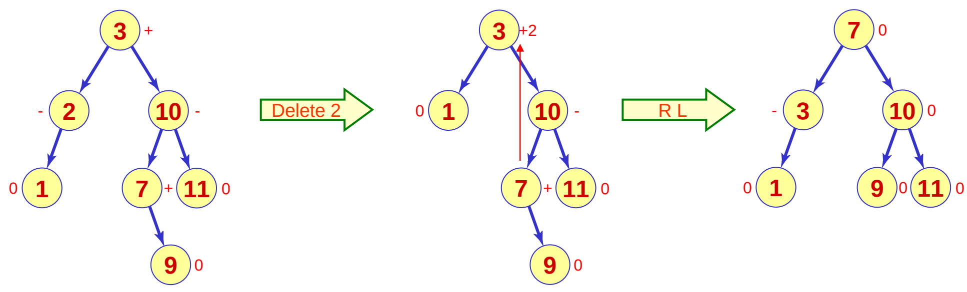
AVL Tree
Examples of the Remove Operation



AVL Tree
Examples of the Remove Operation



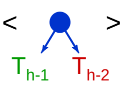
AVL Tree
Examples of the Remove Operation



AVL Tree
The Remove Operation -- Performance

- While insertion of a single key may require at most one rotation, removal of a single key may require a rotation at every node along the search path.
- The worst case is to remove the rightmost node of a Fibonacci tree.
- The rebalancing operation is O(1)
- Tests show that a rotation is required for every five deletions.
- Use AVL trees only if **searching** is more frequent than insertion or deletion.

AVL Tree
The Fibonacci Tree

- Fibonacci trees are the worst case of AVL trees.
- Definition:
  1. The empty tree is the Fibonacci tree of height 0.
  2. A single node is the Fibonacci tree of height 1.
  3. If T_{h-1} and T_{h-2} are Fibonacci trees of heights h-1 and h-2, respectively, then T_h = < 

     > is a Fibonacci tree of height h.
  4. No other trees are Fibonacci trees.

AVL Tree

Example Fibonacci Trees