Data Structures
Chapter 5 Tree

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

우리는 그가 만드신 바라 그리스도 예수 안에서 선한 일을 위하여 지으심을 받은 자니 이 일은 하나님이 전에 예비하사 우리로 그 가운데서 행하게 하려 하심이니라 (엡2:10)

For we are God's workmanship, created in Christ Jesus to do good works, which God prepared in advance for us to do. Eph2:10

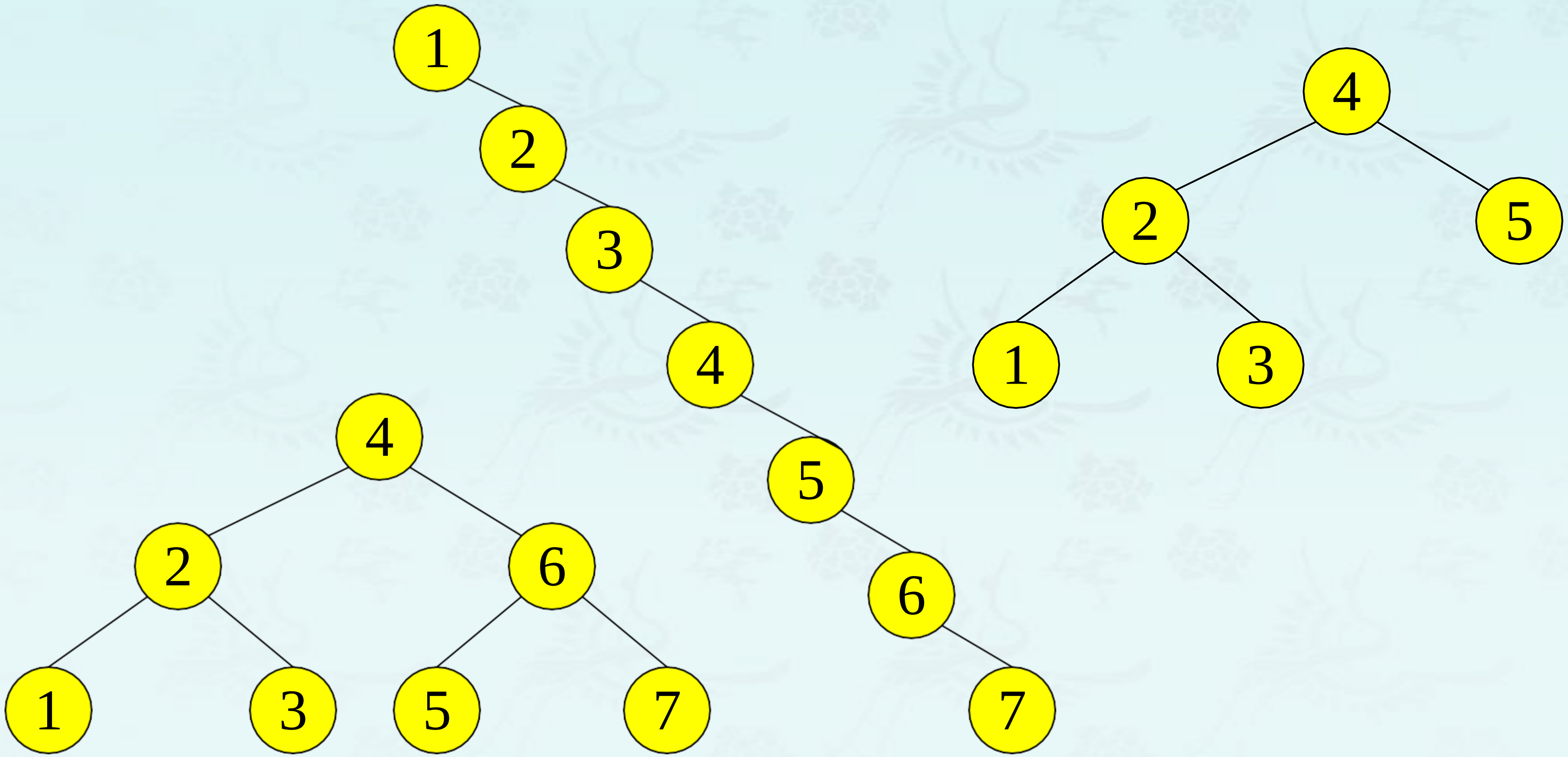
Binary search trees – Revisit

- The time complexity for all BST operations are O(h), where h is tree height (or depth)
- Minimum h is $h = \lfloor \log_2 N \rfloor$ for a binary tree with N nodes
  - What is the best case tree?
  - What is the worst case tree?
- So, **best case** running time of BST operations is $O(\log_2 N)$

- **Worst case** running time is O(N)
  - What happens when you Insert elements in ascending order?
    - Insert: 2, 4, 6, 8, 10, 12 into an empty BST
  - Problem: Lack of "balance";
    - compare depths of left and right subtree
  - Unbalanced degenerate tree

Binary search trees – Revisit

- Balanced and unbalanced BST



Approaches to balancing trees

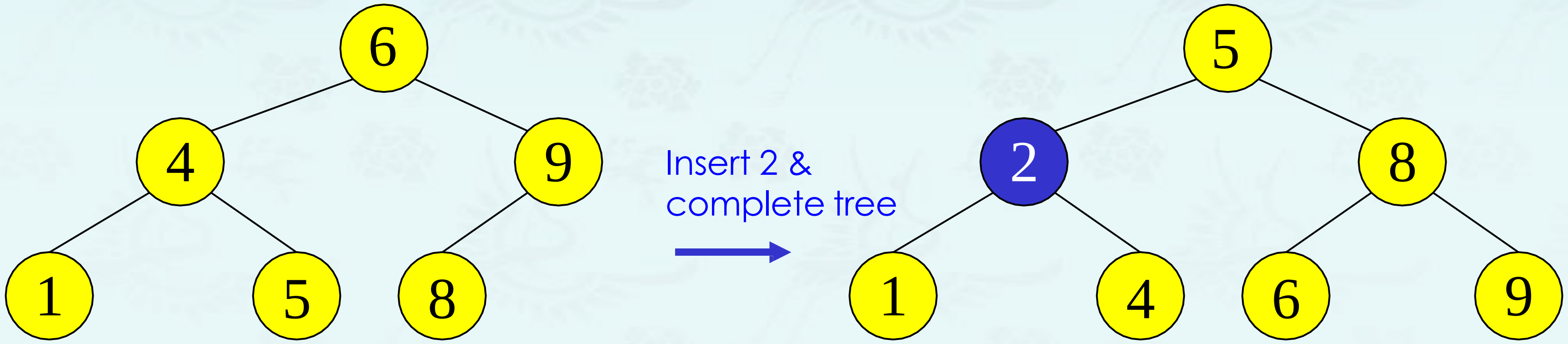
- Don't balance
  - May end up with some nodes very deep
- Strict balance
  - The tree must always be balanced perfectly
- Pretty good balance
  - Only allow a little out of balance
- Adjust on access
  - Self-adjusting

Balancing Binary Search Trees

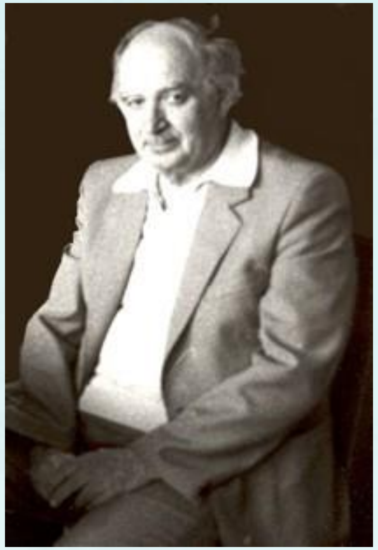
- Many algorithms exist for keeping BST balanced
  - **A**delson-**V**elskii and **L**andis (AVL) trees - (height-balanced trees)
  - Weight-balanced trees
  - **Red-black** trees;
  - **Splay** trees and other self-adjusting trees
  - **B-trees** and other (e.g. 2-4 trees) multiway search trees

Perfect Balance

- Let us suppose we want **a complete tree** after every operation.
  - CBT: The tree is full except possibly in the lower right
- This is expensive.
  - For example, insert 2 in the tree on the left and then rebuild as a complete tree



AVL Tree - Good but not Perfect Balance



AVL Tree (1962)

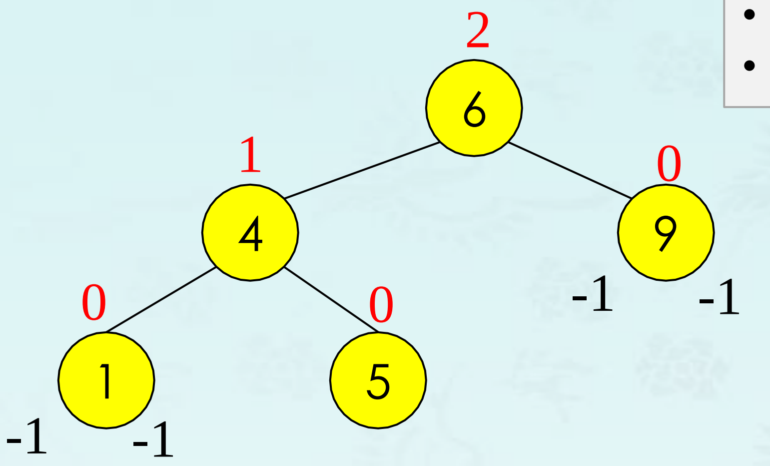
- Named after 2 Russian mathematicians
- Georgii **A**delson-**V**elsky (1922 - 2014)
- Evgenii Mikhailovich **L**andis (1921-1997)

- Height-balanced binary search trees
- Balance factor of a node
  - height(left subtree) - height(right subtree)
- For every node, heights of left and right subtree can differ **by no more than one.**
  - Store current heights in each node or compute it on the fly

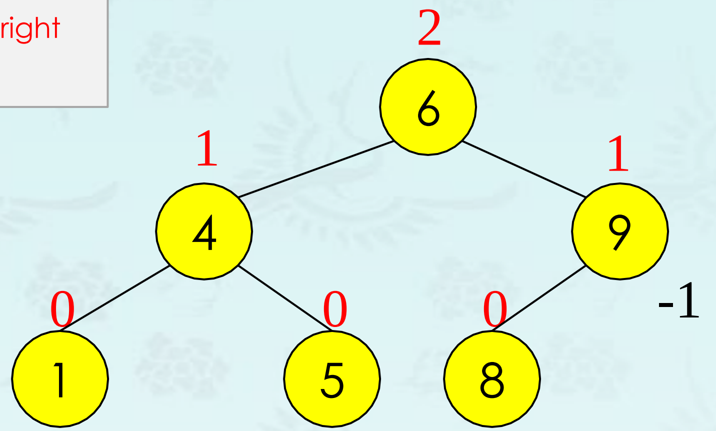
AVL - Node Heights

- height of node = h
- balance factor = h_{left} - h_{right}
- empty height = -1

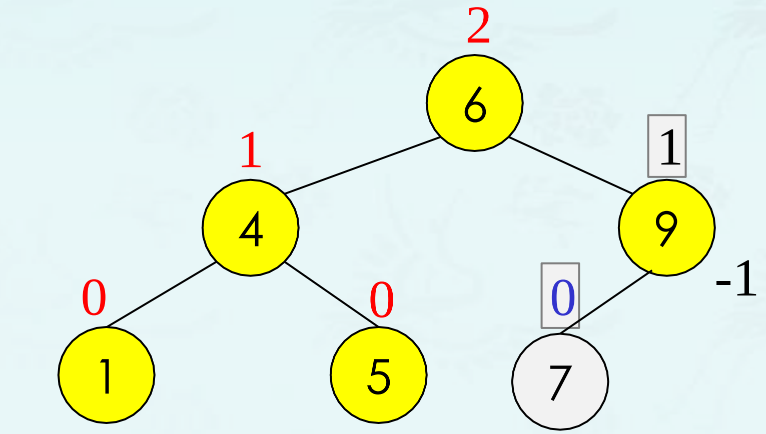
Tree A (AVL)



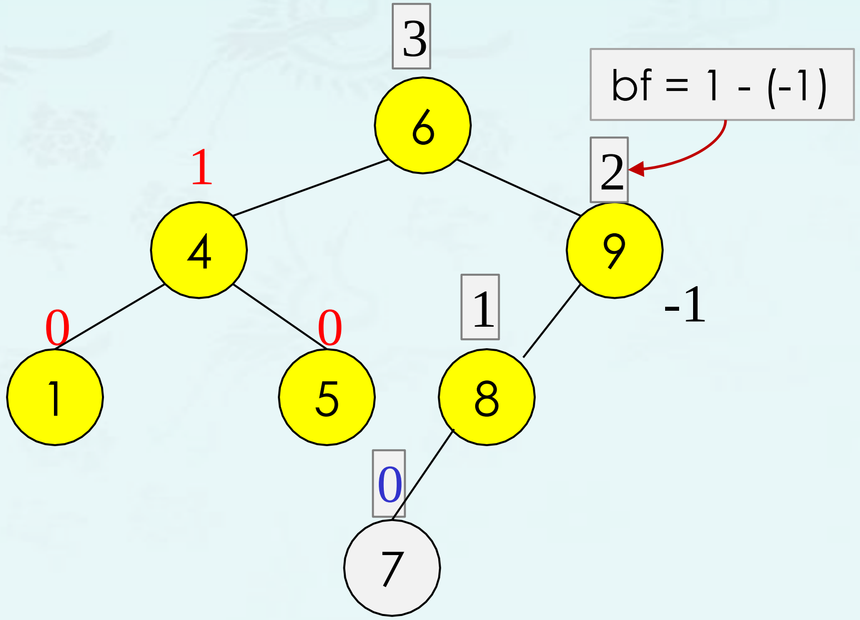
Tree B (AVL)



Node heights
before inserting 7



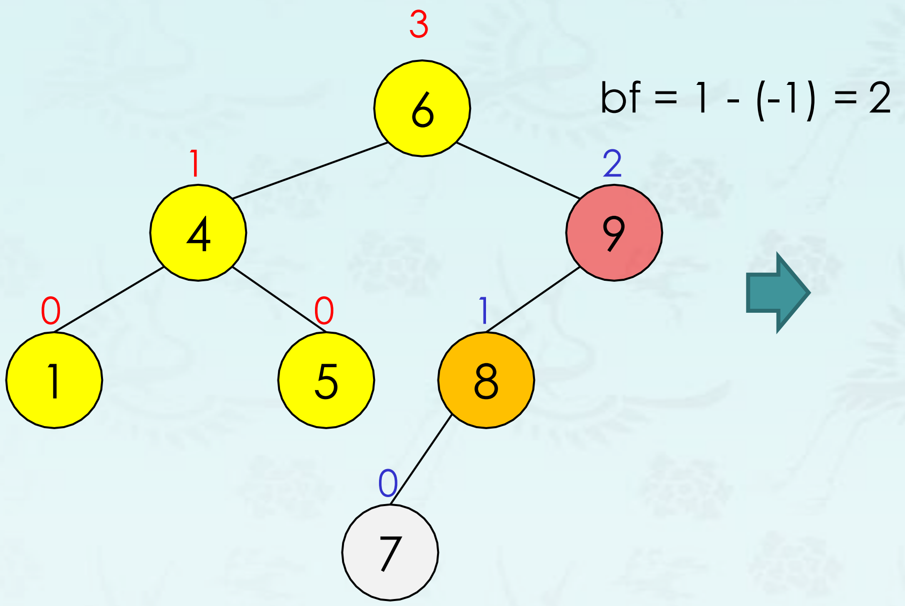
Node heights
after inserting 7



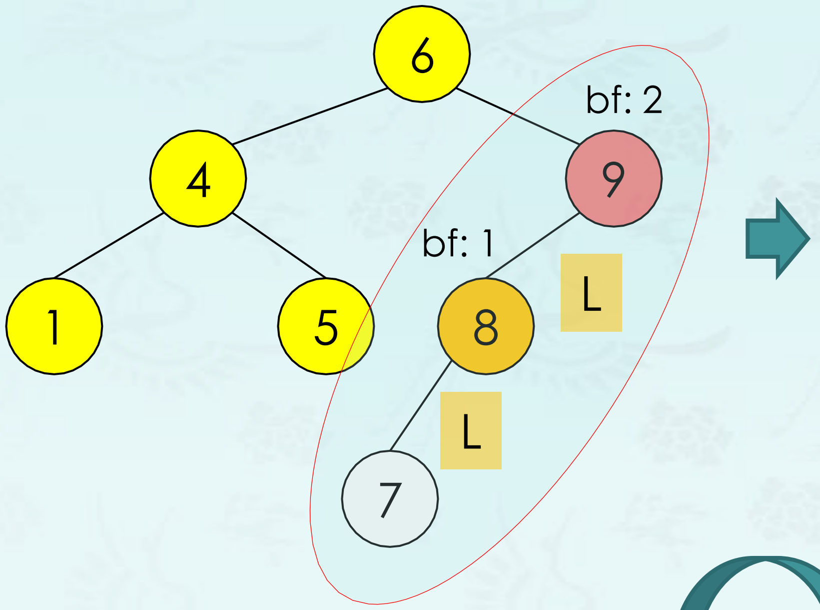
Insert and Rotation in AVL Trees

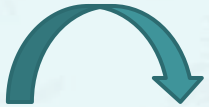
- Insert operation may cause balance factor to become 2 or –2 for some node
  - Only nodes on the path from insertion point to root node have possibly changed in height
  - So after the Insert, **go back up** to the root node by node.
  - If a new balance factor (the difference $h_{left} - h_{right}$) is **2 or –2**, adjust tree by **rotation** around the node

Single Rotation in an AVL Tree



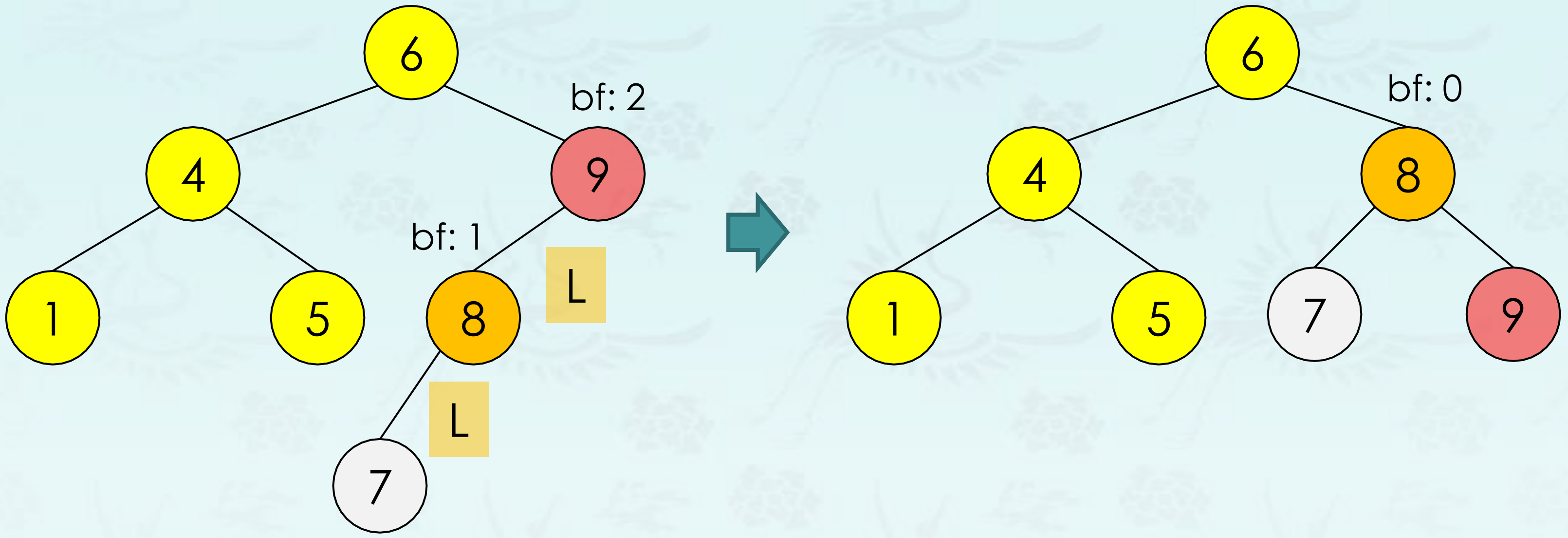
Single Rotation in an AVL Tree





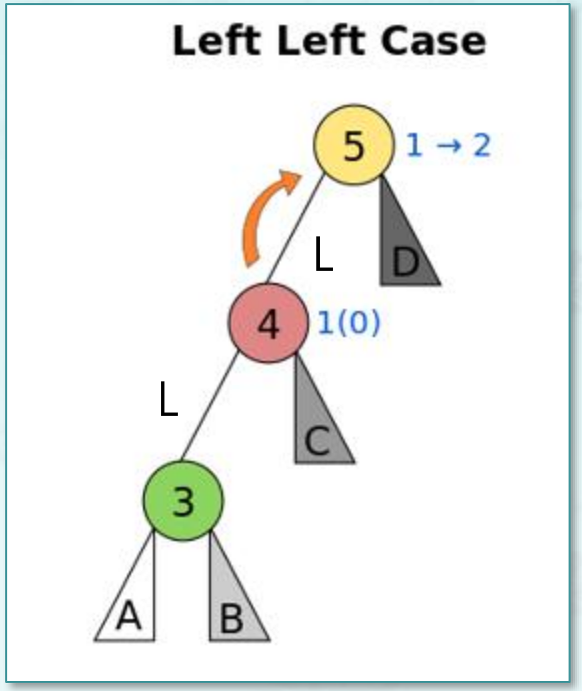
LL case
Rotate right at 9

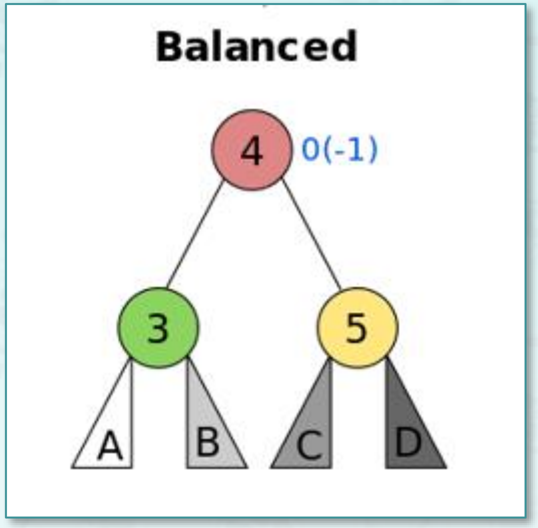
Single Rotation in an AVL Tree



LL case
Rotate right at 9

Single Rotation in an AVL Tree

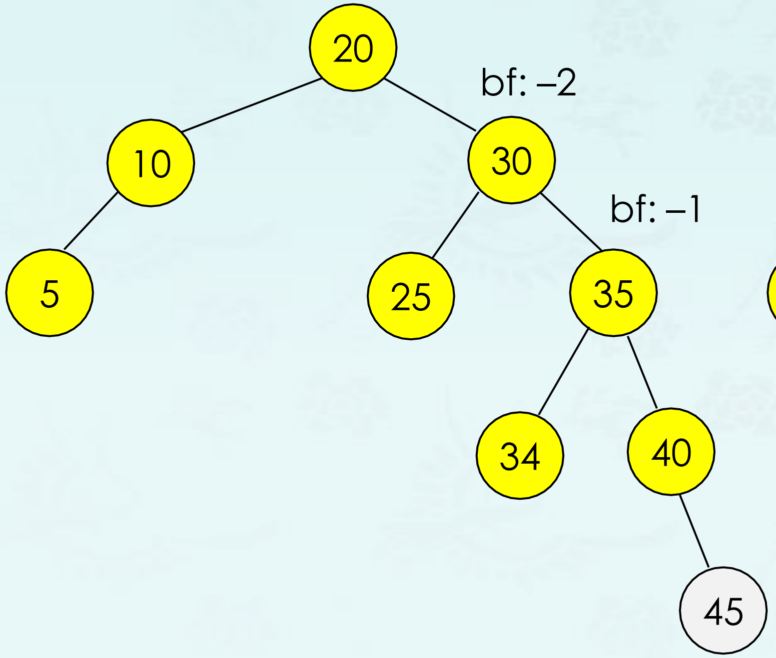




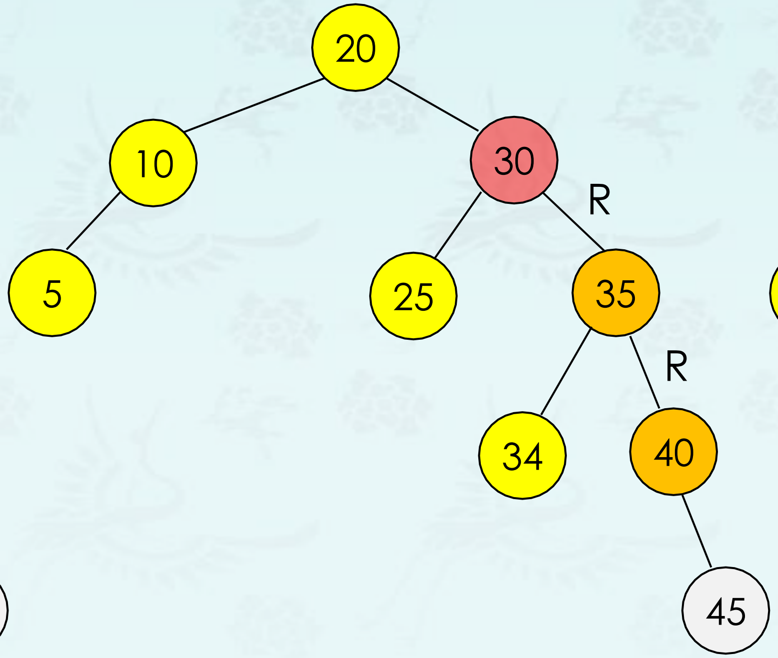
LL Case
Single Right Rotation

AVL Tree Balanced?

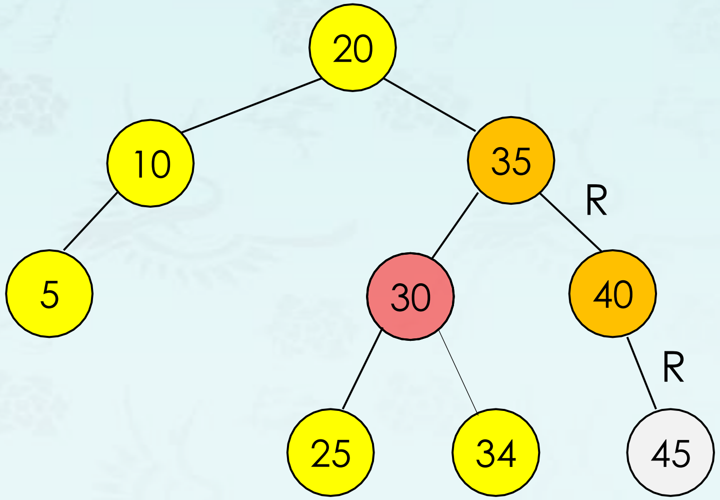
AVL tree balanced
after adding 45?



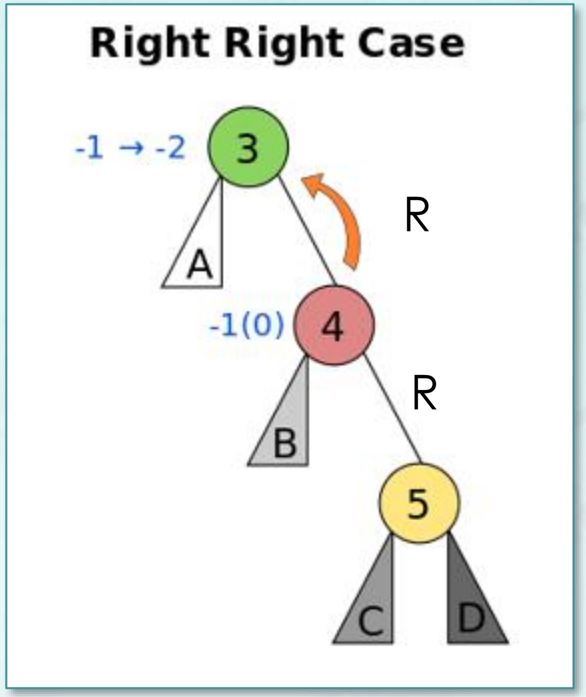
RR case
Rotate left at 30

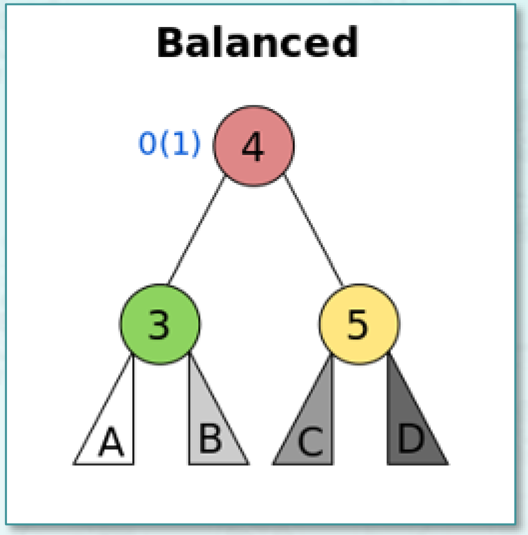


AVL tree balanced
after adding 45



Single Rotation in an AVL Tree

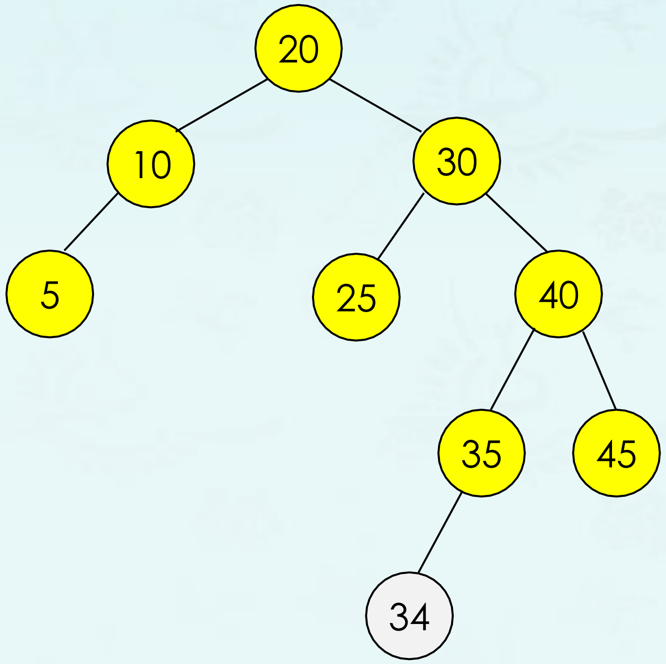




RR Case
Single Left Rotation

AVL Tree Balanced?

- **Insertion of 34**
- Imbalance at ?
- Balance factor ?

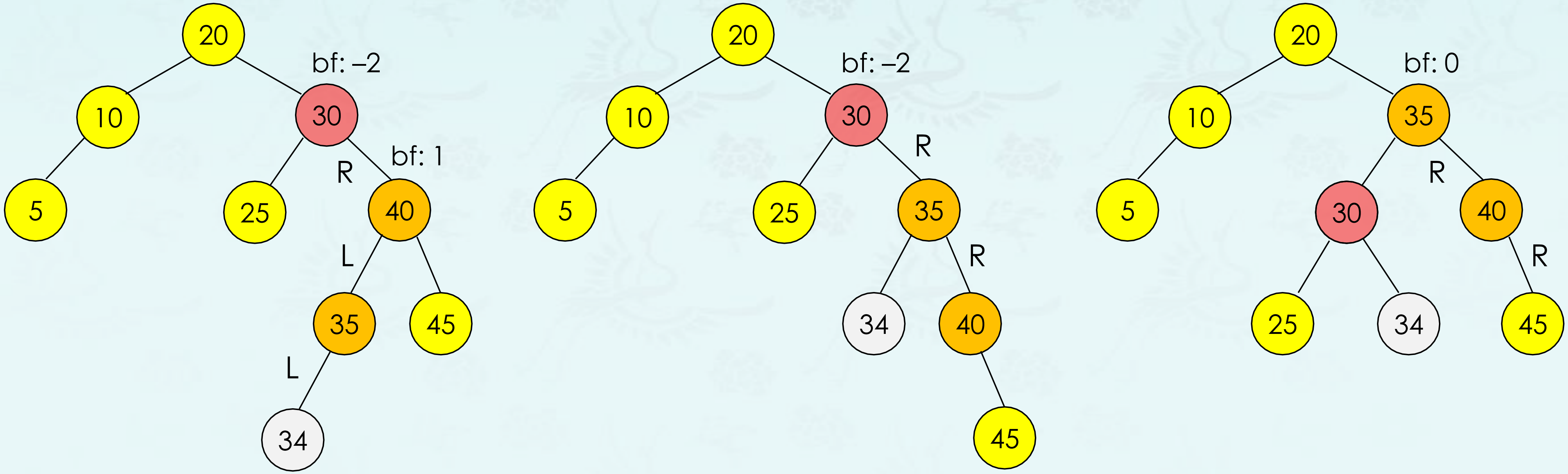


Double rotation RL case

- **Insertion of 34**
- Imbalance at 30
- Balance factor – 2

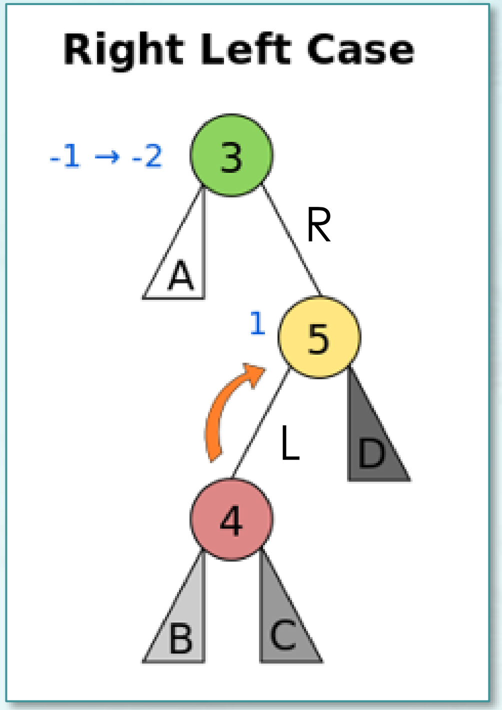
- **RL case** (RR + LL cases)
  - Rotate at 40, LL case
  - Rotate at 30, RR case

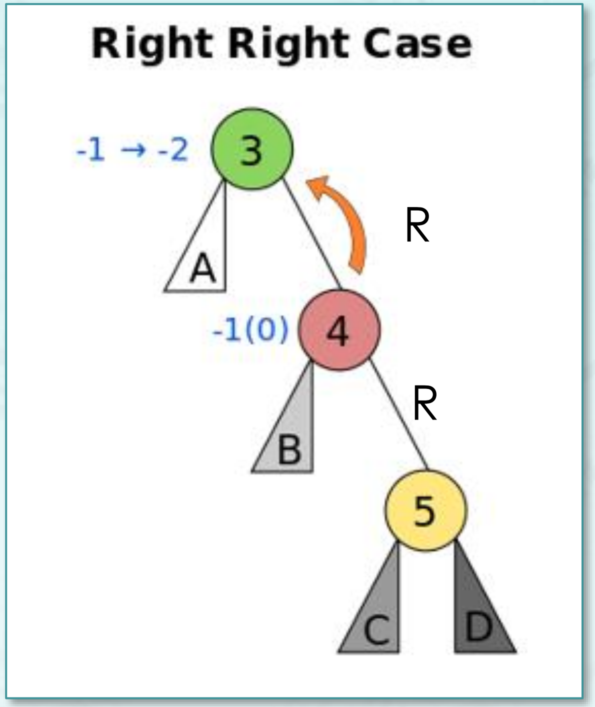
Double rotation

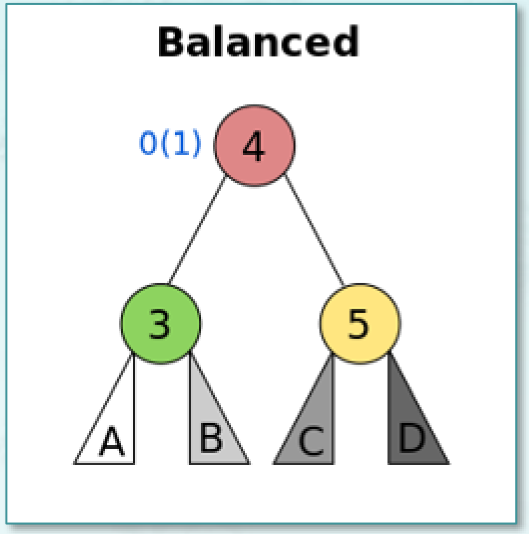


Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

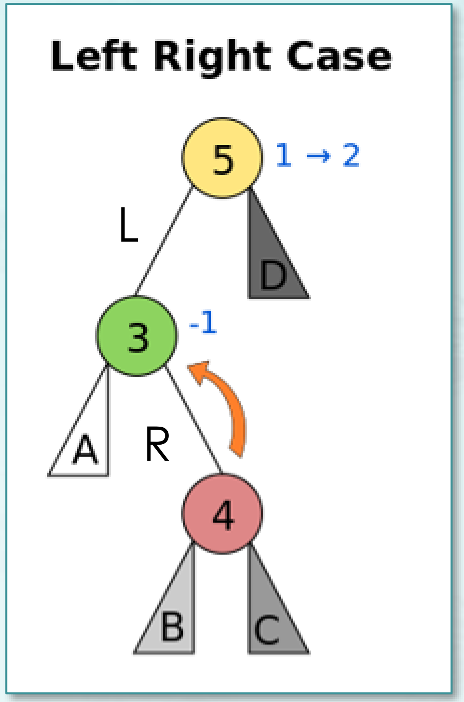
Double rotation – RL Case

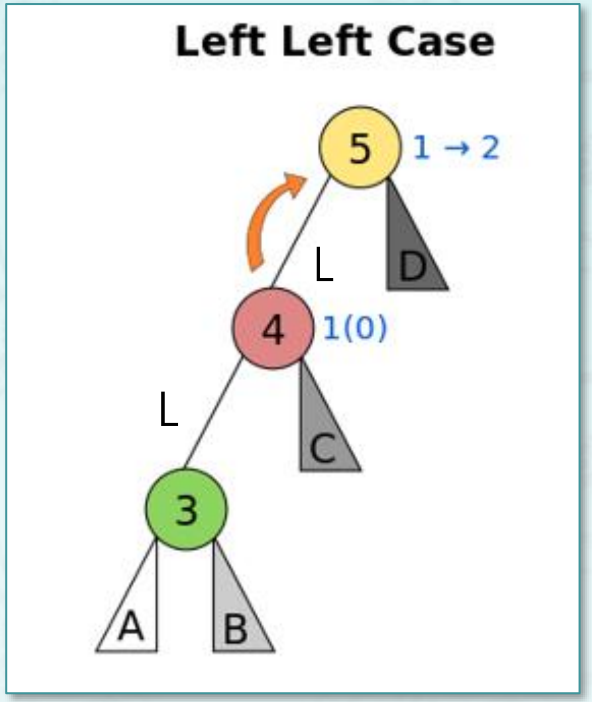


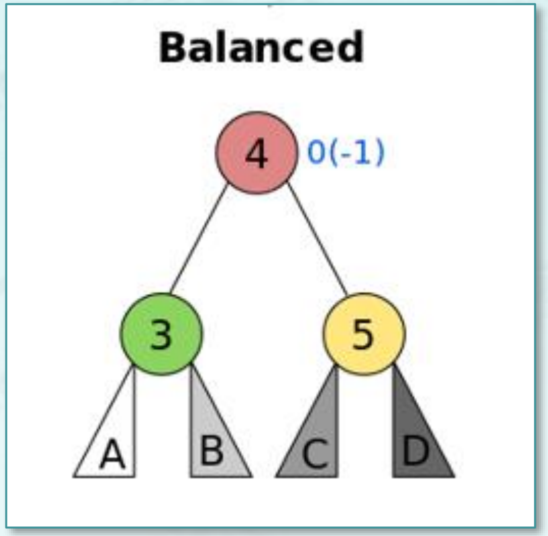




Double rotation – LR Case





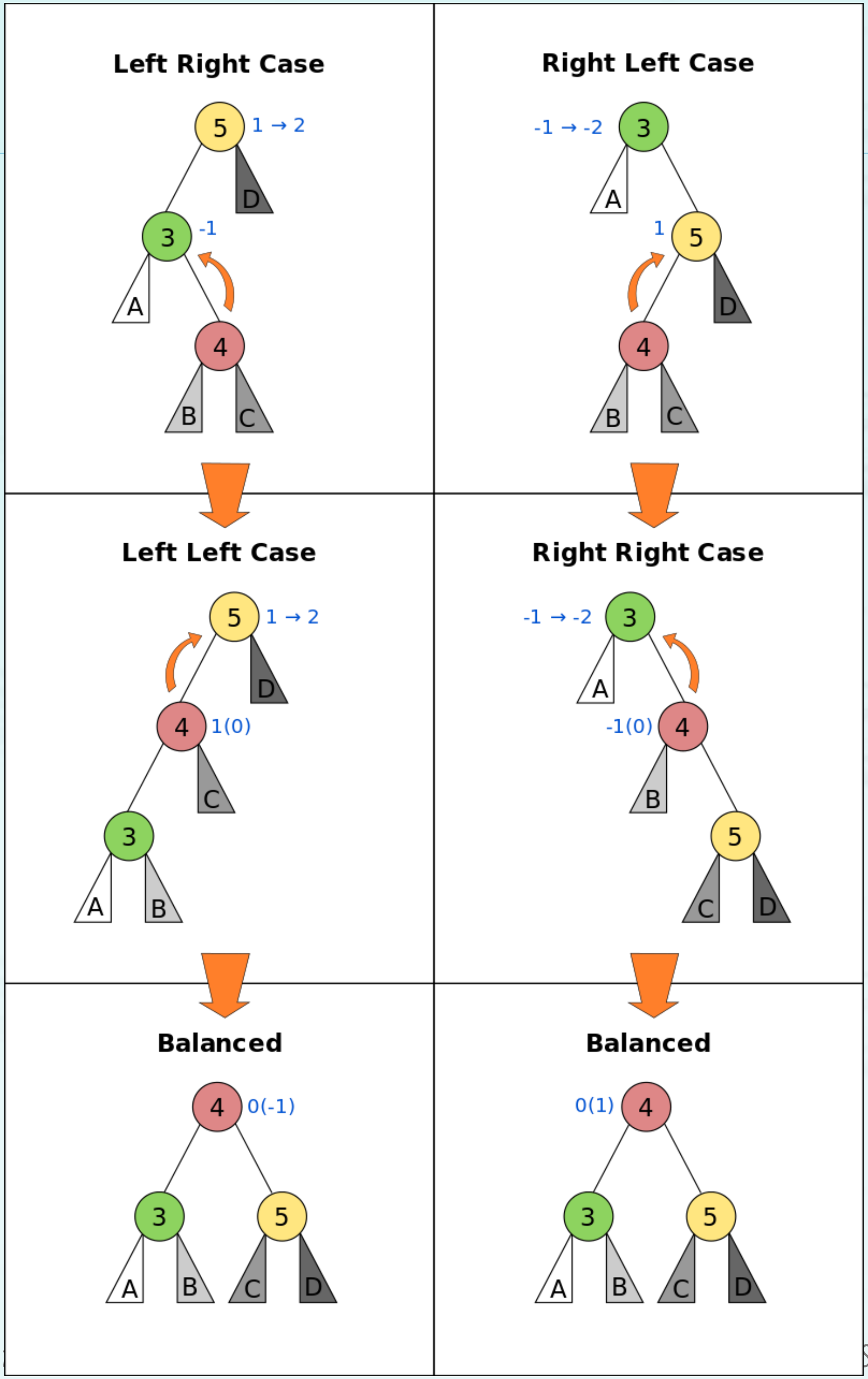


Insertions in AVL Trees

Let the node that needs rebalancing be a.

There are 4 cases:

- Outside Cases (require single rotation) :
  1. Insertion into left subtree of left child of a.
  2. Insertion into right subtree of right child of a.
- Inside Cases (require double rotation) :
  1. Insertion into right subtree of left child of a.
  2. Insertion into left subtree of right child of a.
- The rebalancing is performed through four separate rotation algorithms.



- The numbered circles represent the nodes being rebalanced.
- The lettered triangles represent subtrees which are themselves balanced AVL trees.
- A blue number next to a node denotes possible balance factors
- (those in parentheses occurring only in case of deletion).
- Source: www.wikipedia.com

Pros and Cons of AVL Trees

- Arguments **for** AVL trees:
  - **Search is O(log n)** since AVL trees are always balanced.
  - **Insertion and deletions are also O(log n)**
  - The height balancing adds no more than a constant factor to the speed of insertion.
- Arguments **against** using AVL trees:
  - **Difficult** to program & debug; more space for balance factor.
  - Asymptotically faster but rebalancing costs time.
  - Most large searches are done in database systems on disk and use other structures (e.g. **B-trees**).
  - May be OK to have O(N) for a single operation if total run time for many consecutive operations is fast (e.g. Splay trees).

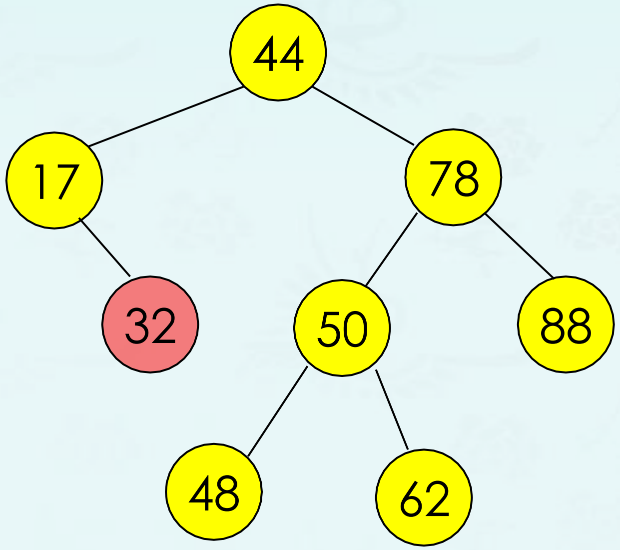
Homework

- Draw AVL trees whenever the tree changes its shape by insertion and deletion. Include trees before and after its rotation and the type of rotation.
- Tree가 모양을 **바꿀 때마다** AVL tree들을 그리고, 각 단계별로 LL, RR, LR, RL을 표시하여 제출하십시오.

(1) Insert the sequence of elements (10, 20, 15, 25, 30, 16, 18, 19) into an AVL tree.
Delete 30 in the AVL tree that you got above and rebalance it.

(2) Delete 32 **in the AVL tree shown below** and rebalance it.

Check your answer with treex.exe.



Data Structures
Chapter 5 Tree

1. Introduction
2. Binary Tree
3. Binary Search Tree
4. **Balancing Tree**
   - **AVL Tree**
   - **Coding**