AVL-Trees

- Introduction
- Insertion
- Deletion

Balance Binary Search Tree

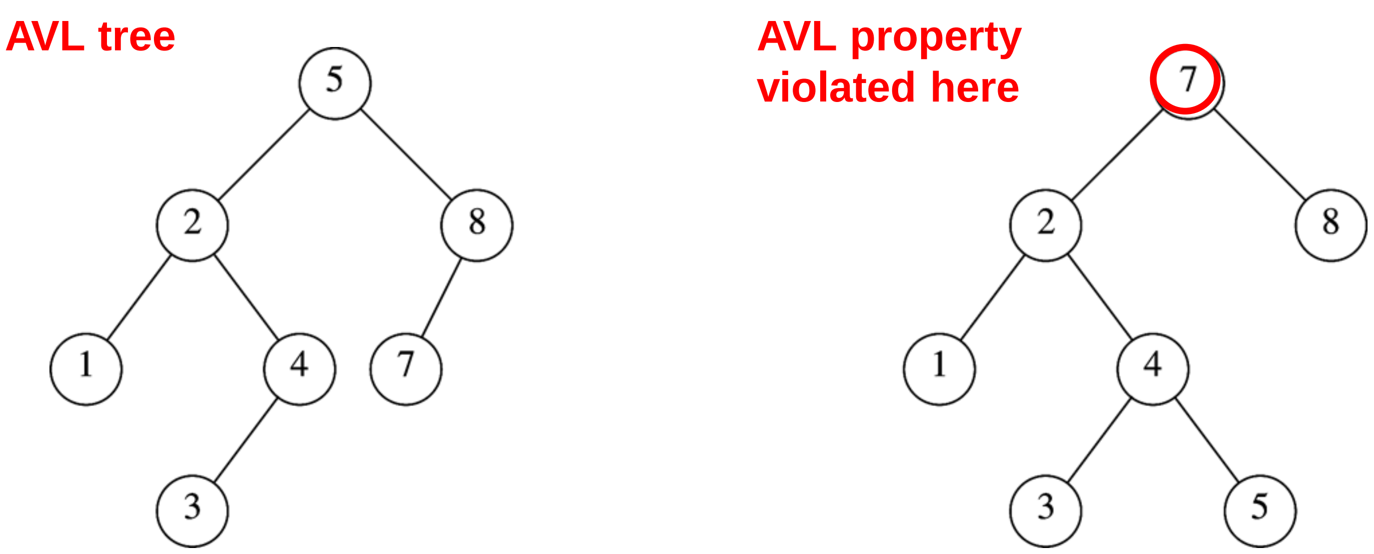
- Worst case height of binary search tree: N
  - Insertion, deletion can be O(N) in the worst case
- We want a tree with small height
- Height of a binary tree with N node is at least $\Theta(\log N)$
- Goal: keep the height of a binary search tree $O(\log N)$
- Balanced binary search trees
  - Examples: AVL tree, red-black tree

Balanced Tree?

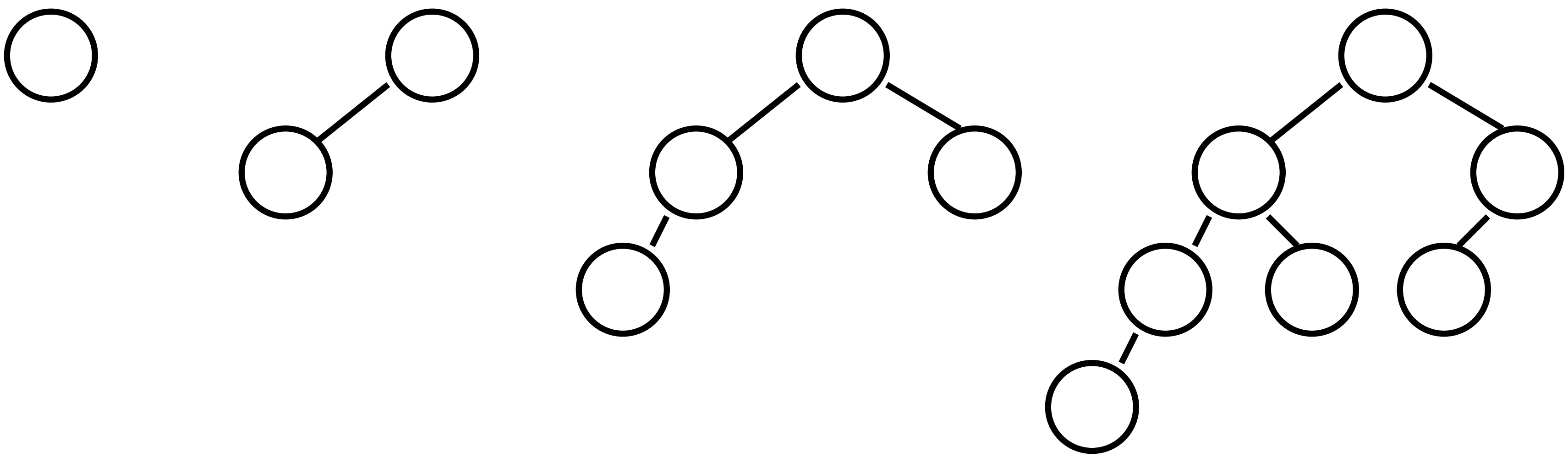
- Suggestion 1: the left and right subtrees of root have the same height
- Suggestion 2: every node must have left and right subtrees of the same height
- Our choice: ***for each node***, the height of the left and right subtrees can differ at most 1,-1,0.

AVL Tree

- An AVL (Adelson-Velskii and Landis 1962) tree is a binary search tree in which
  - for *every* node in the tree, the height of the left and right subtrees differ by at most 1.



AVL Tree with Minimum Number of Nodes



$N_1 = 1$ $N_2 = 2$ $N_3 = 4$ $N_4 = N_2 + N_3 + 1 = 7$

Height of AVL Tree

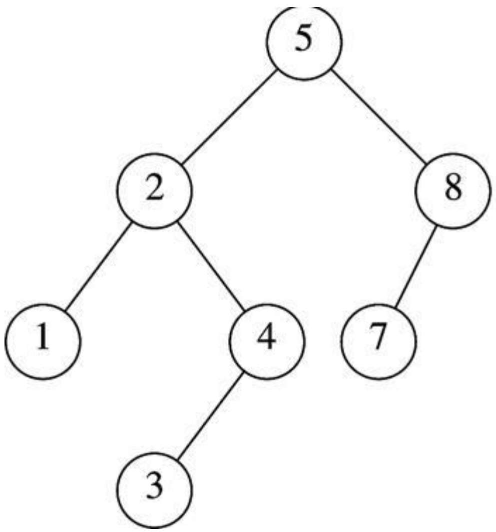
- Denote $\mathbf{N_h}$ the minimum number of nodes in an AVL tree of height h

- $\mathbf{N_1=1,\ N_2=2}$ (base)
  $\mathbf{N_h= N_{h-1} + N_{h-2} +1}$ (recursive relation)

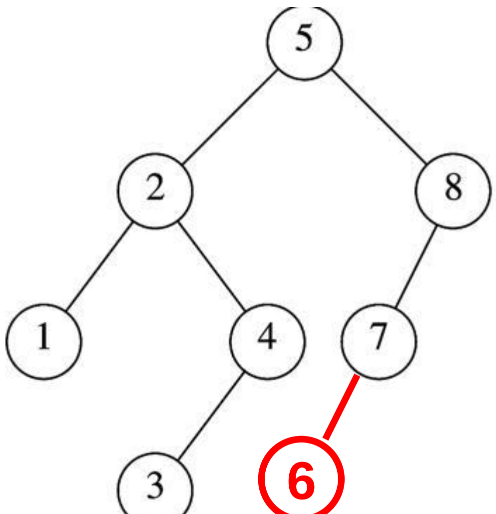
- many operations (i.e. searching) on an AVL tree will take O(log N) time

Insertion in AVL Tree

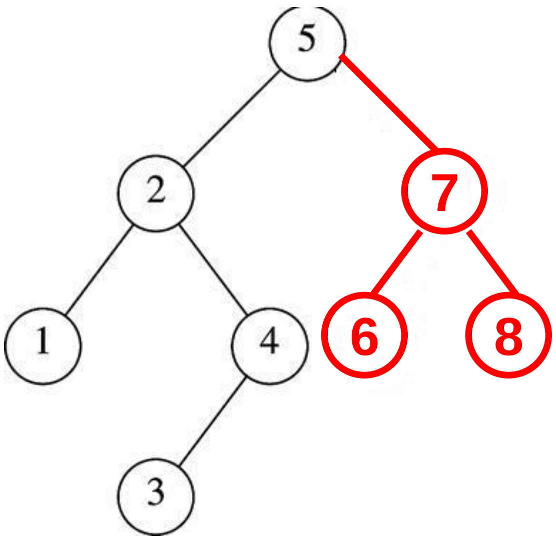
- Basically follows insertion strategy of binary search tree
  - But may cause violation of AVL tree property
- Restore the destroyed balance condition if needed



Original AVL tree



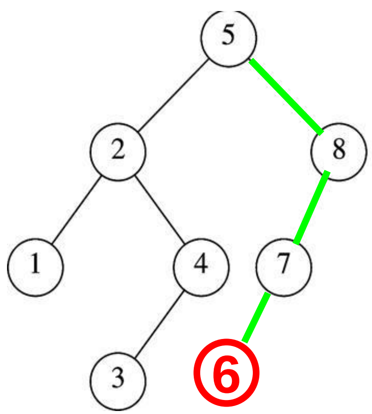
Insert 6
Property violated



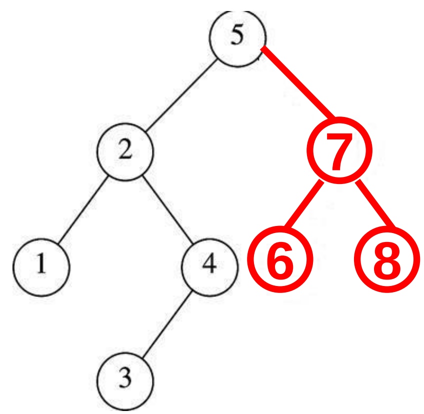
Restore AVL property

Some Observations

- After an insertion, only nodes that are on the path from the insertion point to the root might have their balance altered
  - Because only those nodes have their subtrees altered
- Rebalance the tree at the deepest such node guarantees that the entire tree satisfies the AVL property



Node 5,8,7 might
have balance altered



Rebalance node 7
guarantees the whole tree be AVL

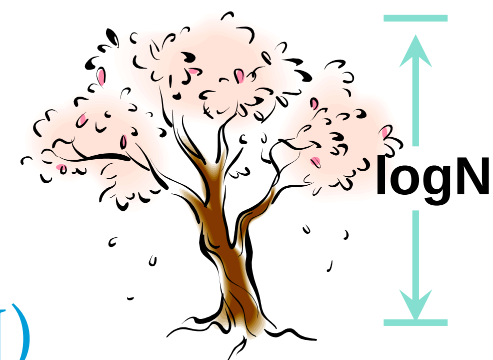
Different Rebalancing Rotations

The rebalancing rotations are classified as LL, LR, RR and RL as illustrated below, based on the position of the inserted node with reference to α.

- **LL rotation:** Inserted node is in the left subtree of the left subtree of node α
- **RR rotation:** Inserted node is in the right subtree of the right subtree of node α
- **LR rotation:** Inserted node is in the right subtree of the left subtree of node α
- **RL rotation:** Inserted node is in the left subtree of the right subtree of node α

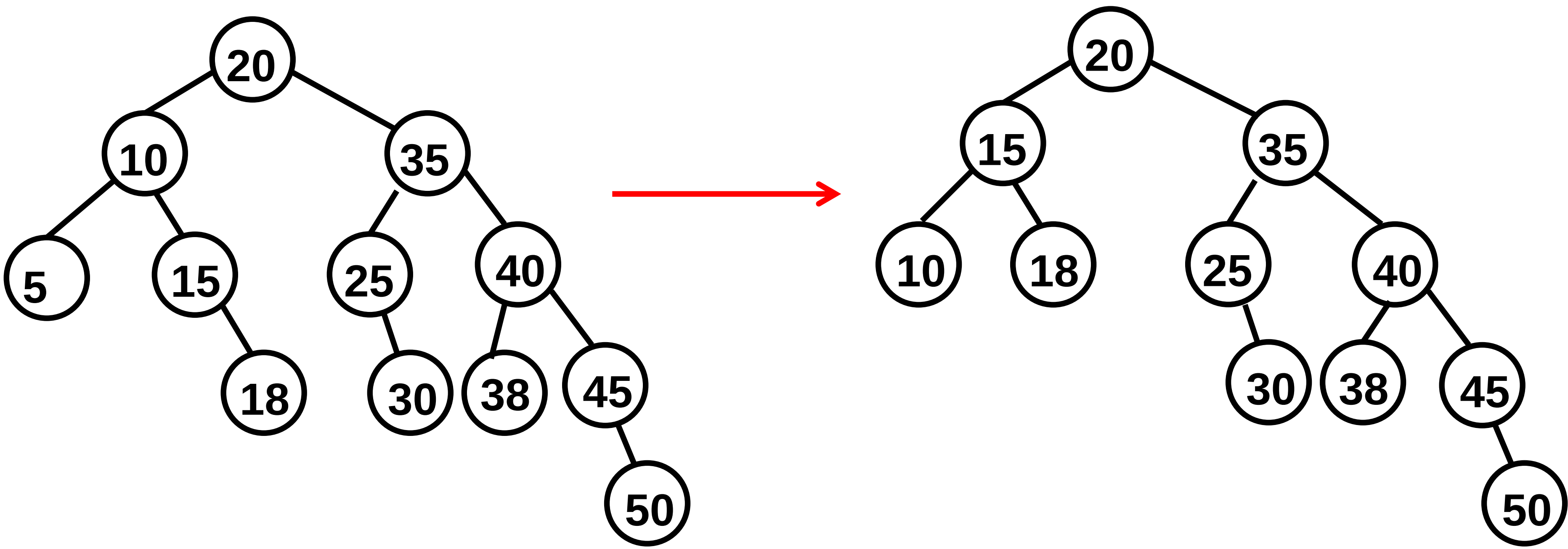
Insertion Analysis

- Insert the new key as a new leaf just as in ordinary binary search tree: O(logN)
- Then trace the path from the new leaf towards the root, for each node x encountered: O(logN)
  - Check height difference: O(1)
  - If satisfies AVL property, proceed to next node: O(1)
  - If not, perform a rotation: O(1)
- The insertion stops when
  - A single rotation is performed
  - Or, we've checked all nodes in the path
- Time complexity for insertion O(logN)



Deletion from AVL Tree

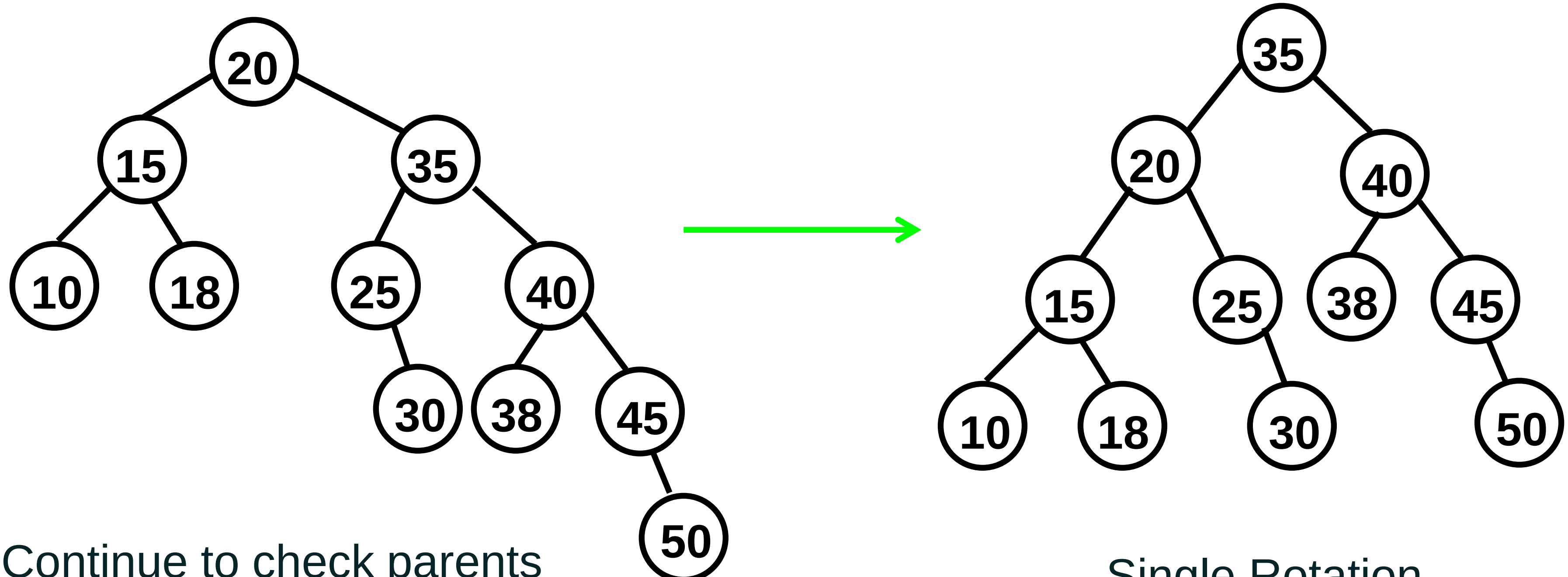
- Basically follows deletion strategy of binary search tree
  - But may cause violation of AVL tree property
- Restore the destroyed balance condition if needed



Delete 5, Node 10 is unbalanced

Single Rotation

Cont'd



Continue to check parents

Oops!! Node 20 is unbalanced!!

Single Rotation

For deletion, after rotation, we need to continue tracing upward to see if AVL-tree property is violated at other node.

Different rotations are classified in R0, R1, R-1, L0, L1 And L-1

Different rotations

- An element can be deleted from AVL tree which may change the BF of a node such that it results in unbalanced tree.
- Some rotations will be applied on AVL tree to balance it.
- R rotation is applied if the deleted node is in the right subtree of node A (A is the node with balance factor other than 0, 1 and -1).
- L rotation is applied if the deleted node is in the left subtree of node A.

Different rotations cont…

- Suppose we have deleted node X from the tree.
- A is the closest ancestor node on the path from X to the root node, with a balance factor -2 or +2.
- B is the desendent of node A on the opposite subtree of deleted node i.e. if the deleted node is on left side the B is the desendent on the right subtree of A or the root of right subtree of A.

R Rotation

- R Rotation is applied when the deleted node is in the right subtree of node A.
- There are three different types of rotations based on the balanced factor of node B.
- R0 Rotation: When the balance Factor of node B is 0.
  - Apply LL Rotation on node A.
- R1 Rotation: When the balance Factor of node B is +1.
  - Apply LL Rotation on node A.
- R-1 Rotation: When the balance Factor of node B is -1.
  - Apply LR Rotation(RR rotation on B and LL rotation on node A).

L Rotation

- L Rotation is applied when the deleted node is in the left subtree of node A.
- There are three different types of rotations based on the balanced factor of node B.
- L0 Rotation: When the balance Factor of node B is 0.
    - Apply RR Rotation on node A.
- L-1 Rotation: When the balance Factor of node B is -1.
    - Apply RR Rotation on node A.
- L1 Rotation: When the balance Factor of node B is +1.
    - Apply RL Rotation(LL rotation on B and RR rotation on node A).

Thank You !!!