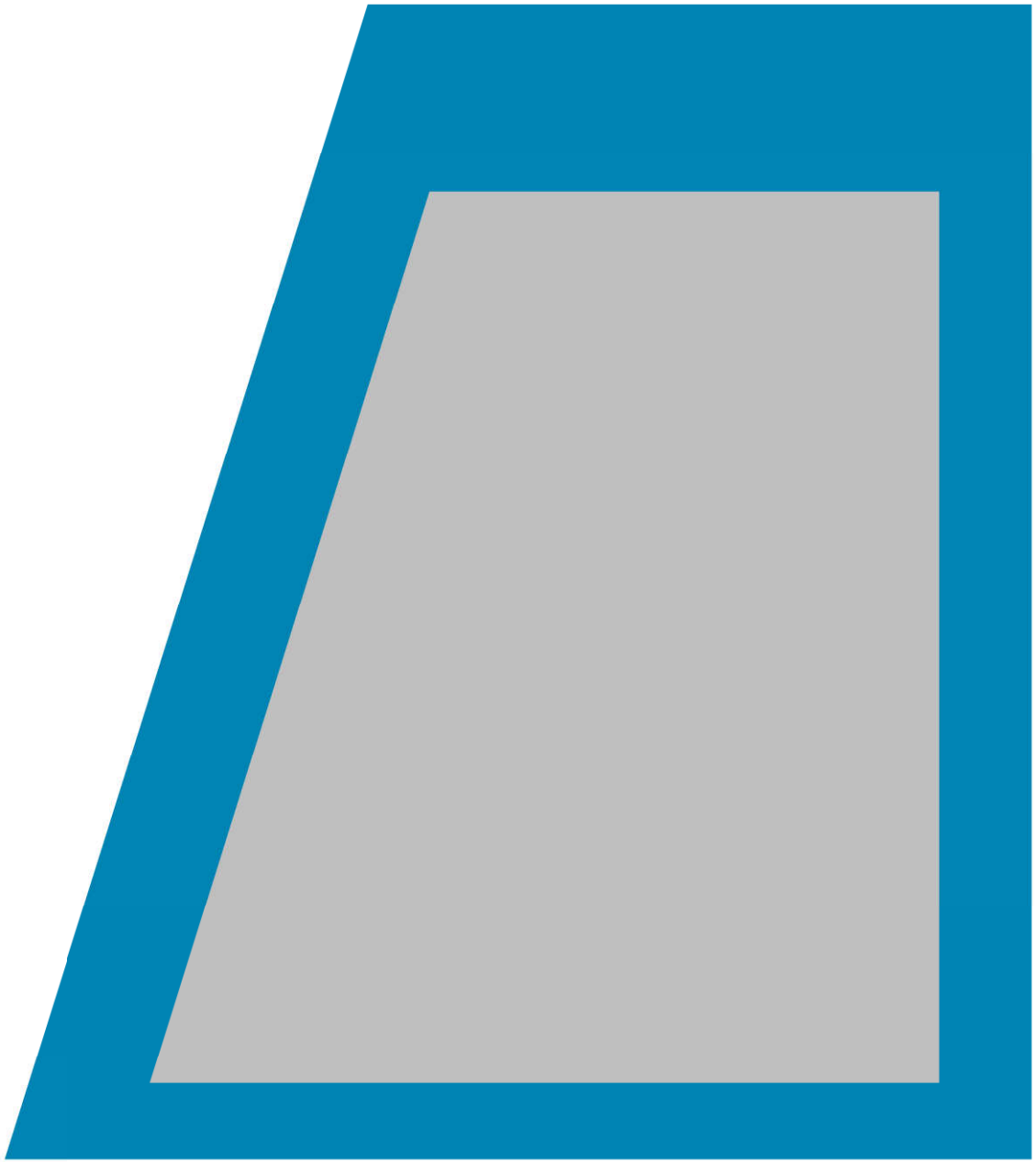
ITD62-124 Data Structure

Balanced Search Tree

Outline:

- Introduction
- Rebalancing
- Single Rotation
- Double Rotation
- Insertion
- Deletion



Introduction

- **AVL** Tree has been proposed by **A**del'son-**V**el'skii and **L**andis
- an AVL tree is a binary search tree such that for any node in the tree, the height of the left and right subtrees can differ by at most 1.

Introduction

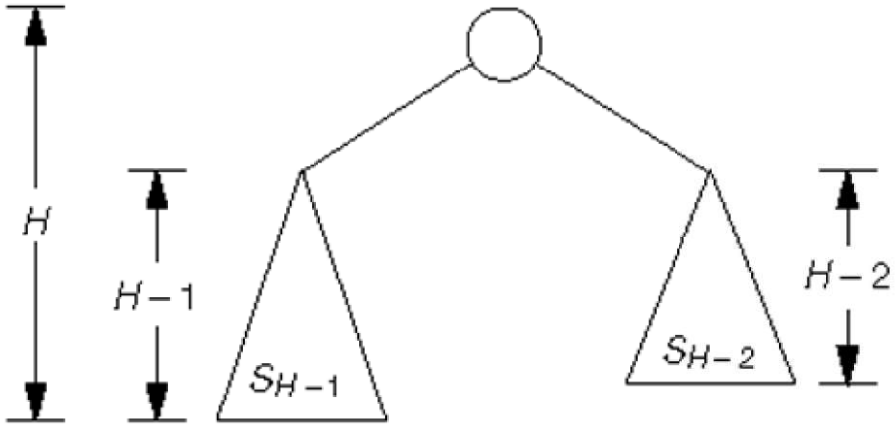
- **AVL** Tree is any tree with these conditions :
  - ✓ must be a binary tree
  - ✓ must be a binary search tree
  - ✓ must be a balanced tree

Introduction

- *Balance factors* are the differences between the heights of the left and right subtrees. For an AVL Tree, all balance factors should be +1, 0, or -1.
- AVL tree *approximates* the ideal tree (completely balanced tree)

Introduction

Minimum tree of height *H*



Introduction

- **Properties:**
  - ✓ An update (insert or delete) in an AVL tree could destroy the balance. It must then be rebalanced before the operation can be considered complete.
  - ✓ After an insertion, only nodes that are on the path from the insertion point to the root can have their balances altered.

Rebalancing

- Suppose the node to be rebalanced is X. There are four cases that we might have to fix :
  - ✓ Case 1: An insertion in the left subtree of the left child of X
  - ✓ Case 2: An insertion in the right subtree of the left child of X
  - ✓ Case 3: An insertion in the left subtree of the right child of X
  - ✓ Case 4: An insertion in the right subtree of the right child of X

Rebalancing

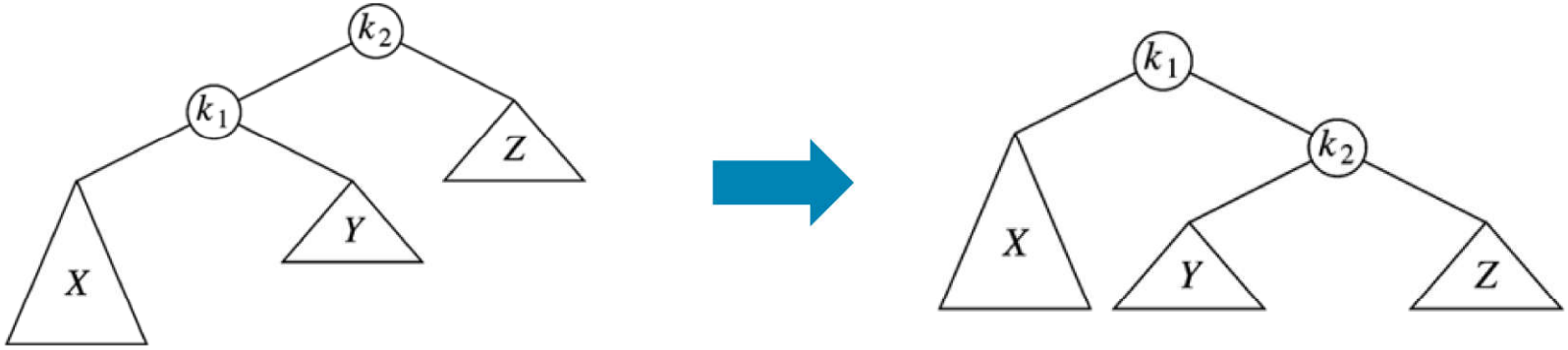
- **Balancing Operations: Rotations**
  - ✓ **Case 1** and **case 4** are symmetric and requires the same operation for balance.
    - ➢ Cases 1 and 4 are handled by *single rotation*
  - ✓ **Case 2** and **case 3** are symmetric and requires the same operation for balance.
    - ➢ Cases 2 and 3 are handled by *double rotation*

Single Rotation

- A single rotation switches the roles of the parent and child while maintaining the search order
- Rotate between a node and its child
  - ✓Child becomes parent
  - ✓Parent becomes right child in case 1, left child in case 4
- The result is a binary search tree that satisfies the AVL property

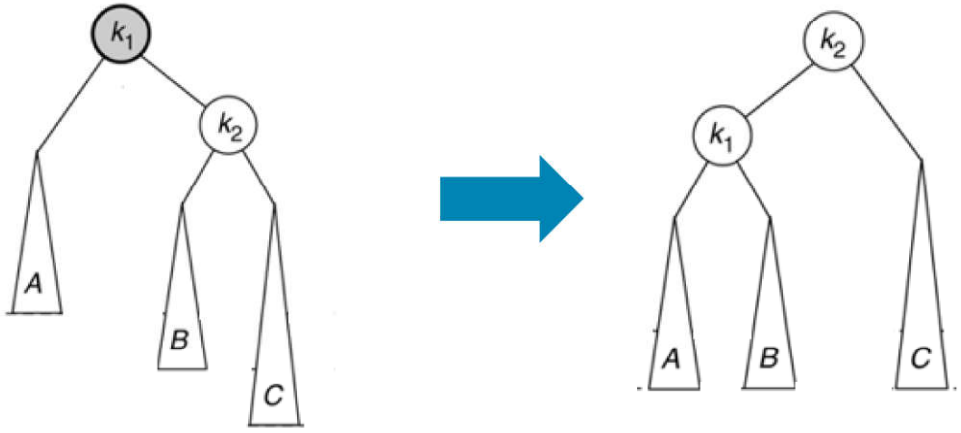
Single Rotation

- **Case 1:**
  - ✓ Replace node k_2 by node k_1
  - ✓ Set node k_2 to be right child of node k_1
  - ✓ Set subtree Y to be left child of node k_2



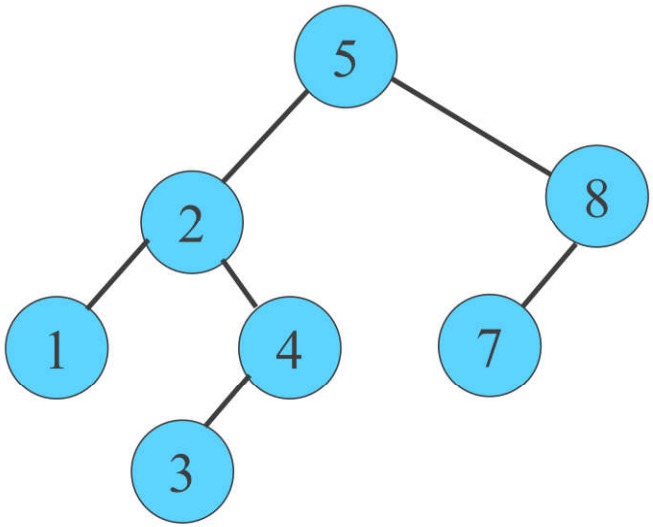
Single Rotation

- **Case 4:**
  - ✓ Replace node k_1 by node k_2
  - ✓ Set node k_1 to be left child of node k_2
  - ✓ Set subtree B to be right child of node k_1



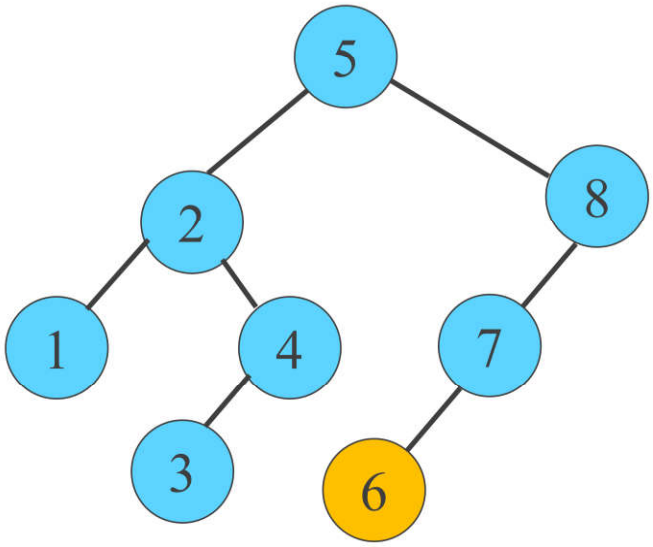
Single Rotation

- Example: Inserting node 6 in the AVL tree



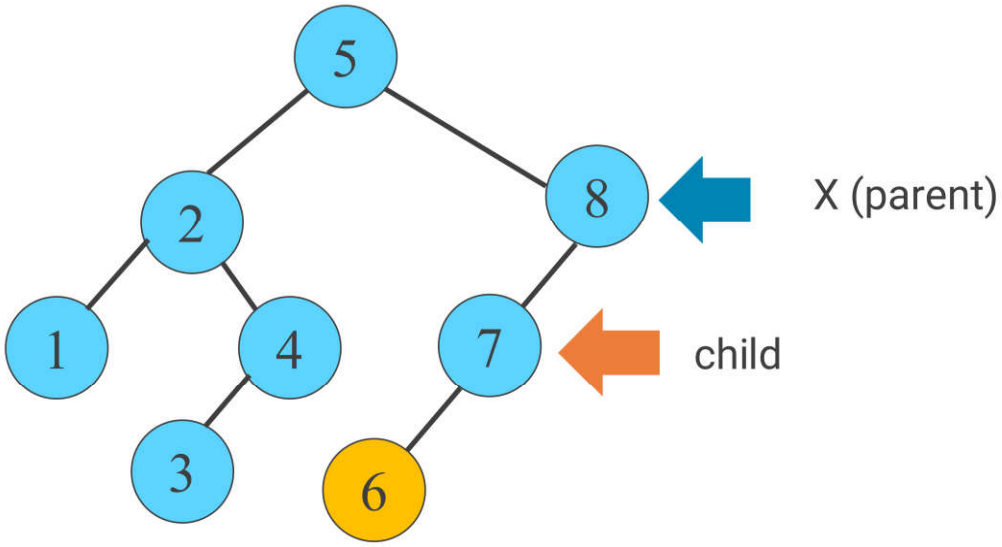
Single Rotation

- Solution:



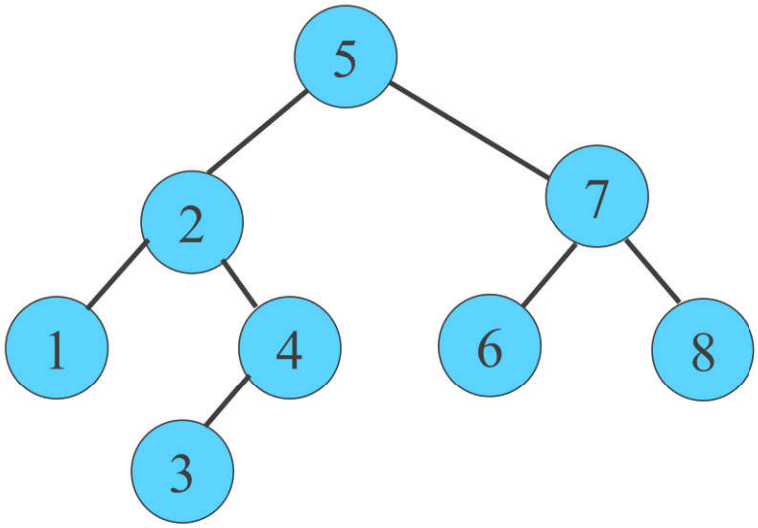
Single Rotation

- Solution:



Single Rotation

- Solution:



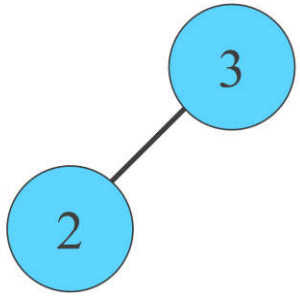
Single Rotation

- Example: Inserting 3, 2, 1, 4, 5, 6 and 7 in an empty AVL tree

Single Rotation

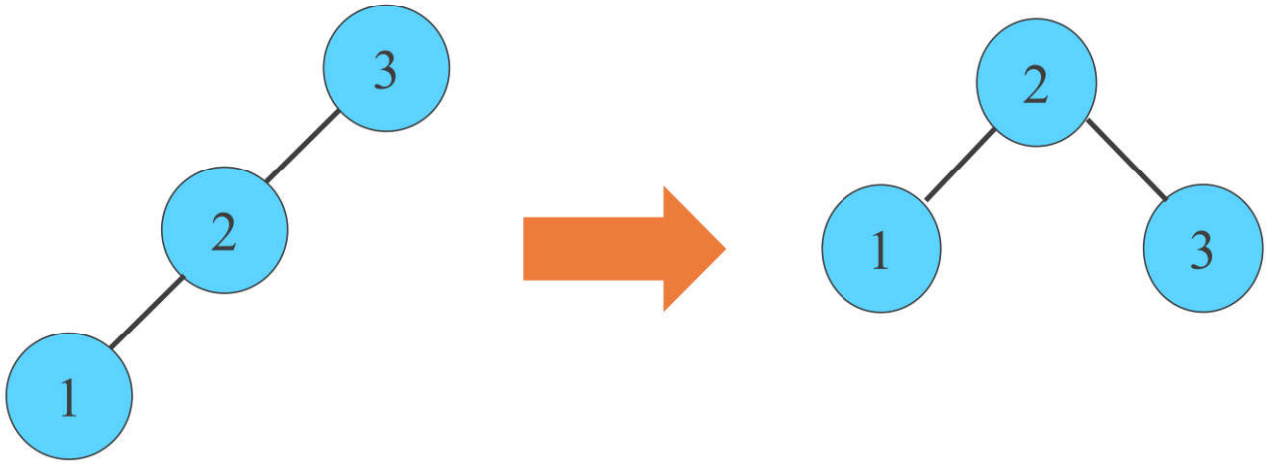
- Solution:
  - ✓inserting 3

  - ✓inserting 2



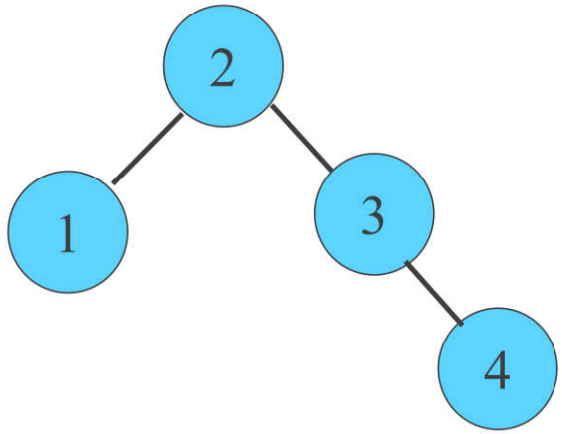
Single Rotation

- Solution:
  - ✓inserting 1



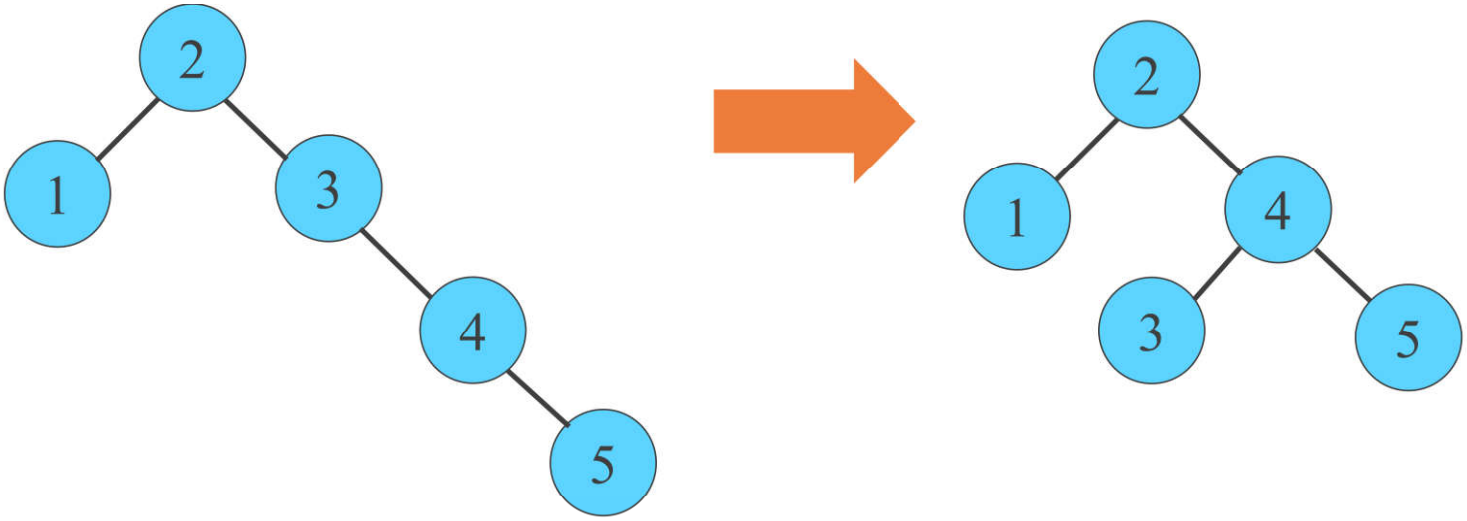
Single Rotation

- Solution:
  - ✓inserting 4



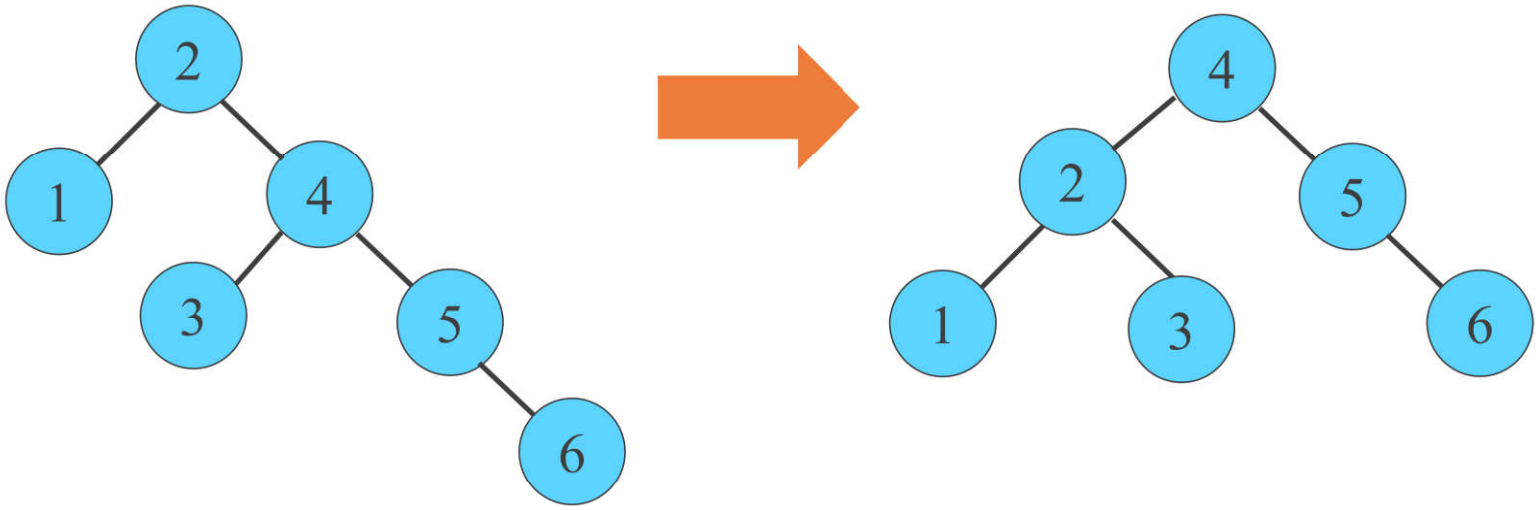
Single Rotation

- Solution:
  - ✓inserting 5



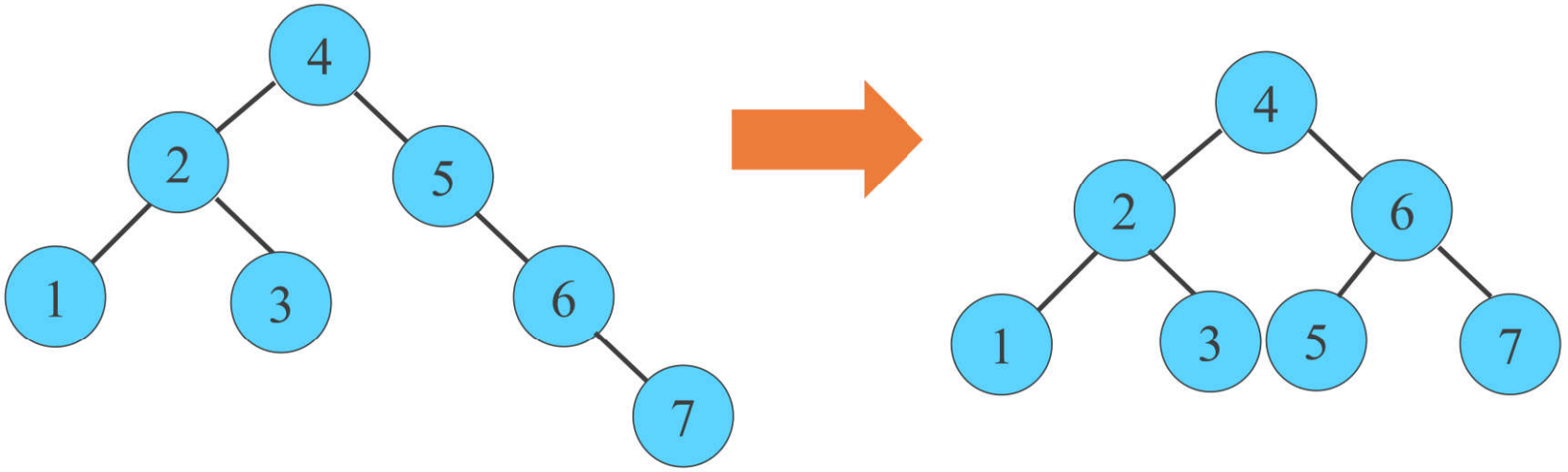
Single Rotation

- Solution:
  - ✓inserting 6



Single Rotation

- Solution:
  - ✓inserting 7

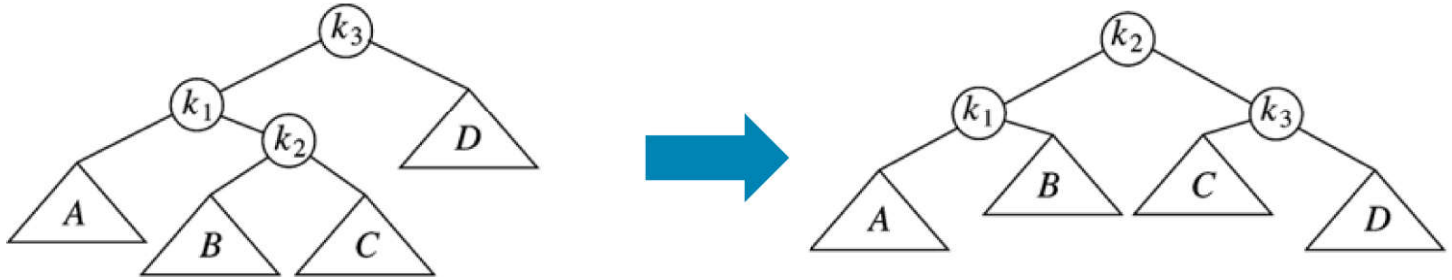


Double Rotation

- A Single rotation does not fix the case 2 and case 3
- These cases require a *double* rotation, involving three nodes and four subtrees

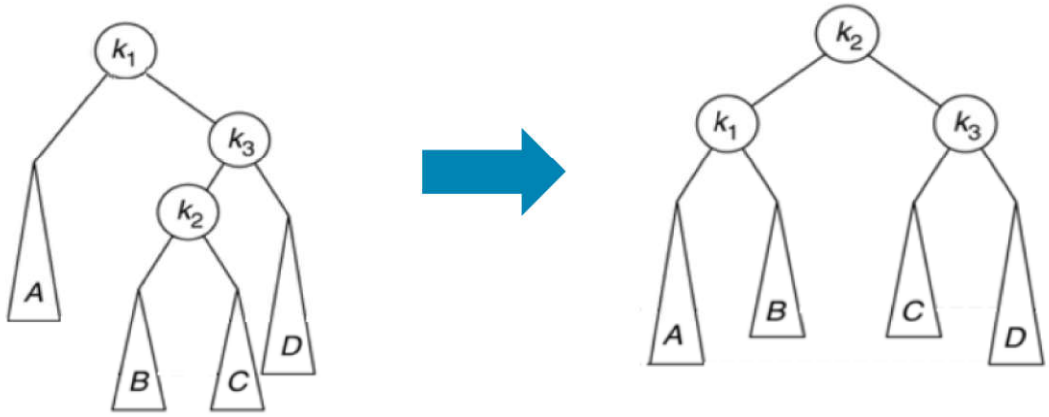
Double Rotation

- **Case 2:**
  - ✓ Left-right double rotation to fix case 2
  - ✓ First rotate between k_1 and k_2
  - ✓ Then rotate between k_2 and k_3



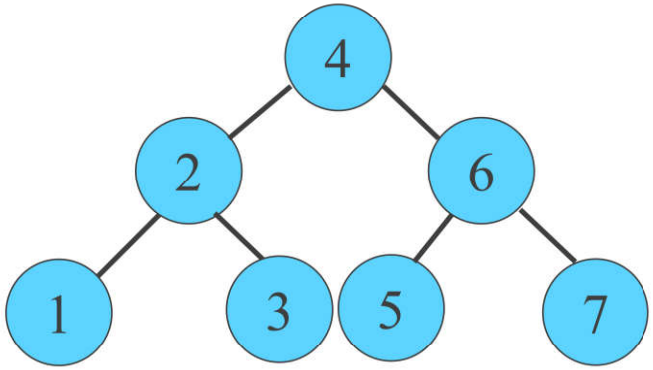
Double Rotation

- **Case 3:**
  - ✓ Right-left double rotation to fix case 3
  - ✓ First rotate between k_2 and k_3
  - ✓ Then rotate between k_2 and k_1



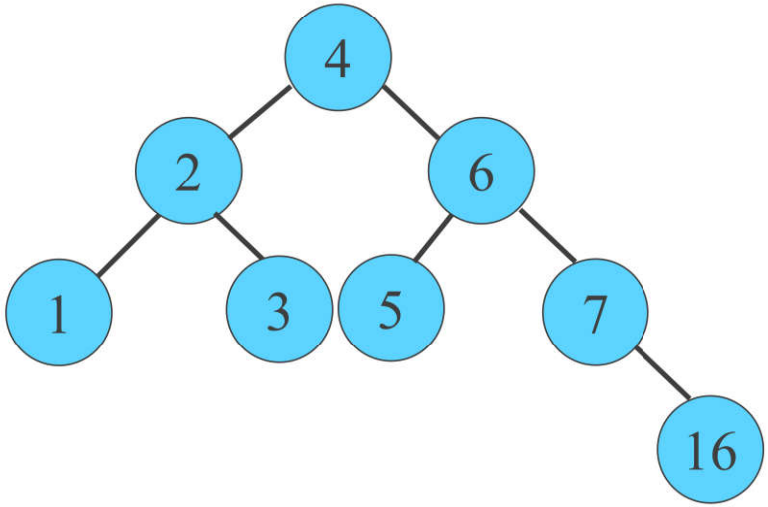
Double Rotation

- **Example:** Inserting 16 and 15 in the AVL tree



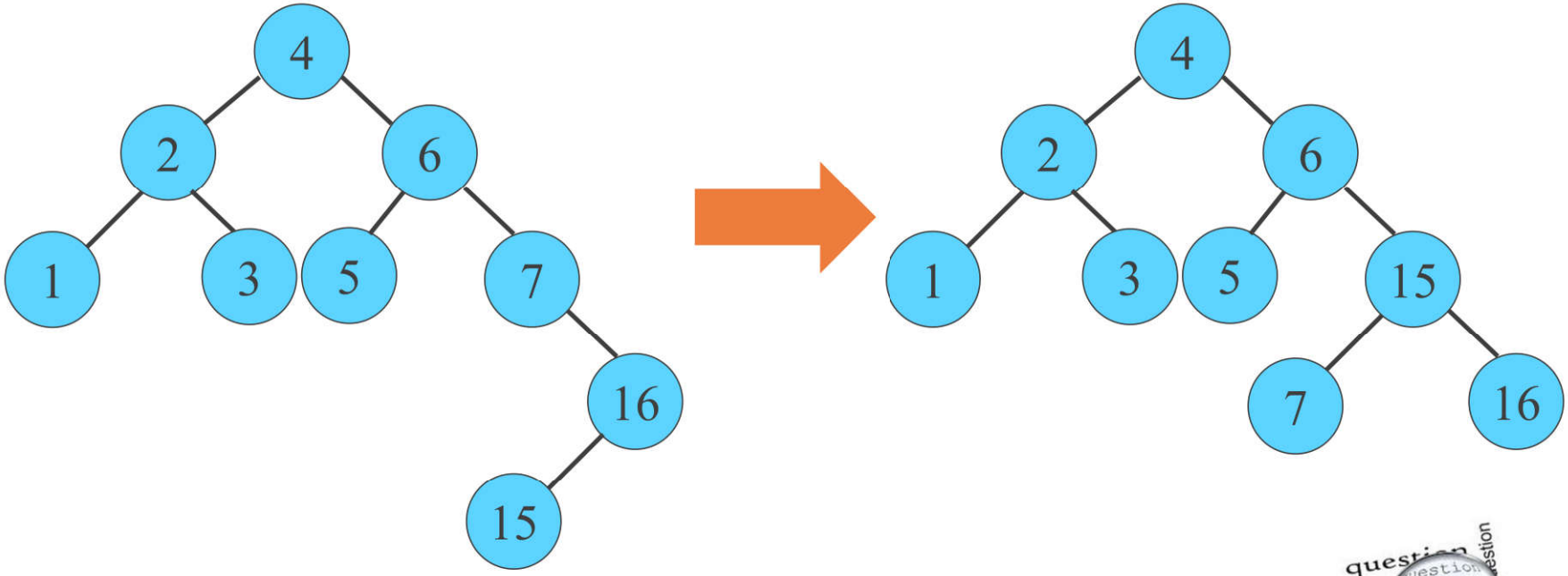
Double Rotation

- **Solution:**
  - ✓Inserting 16



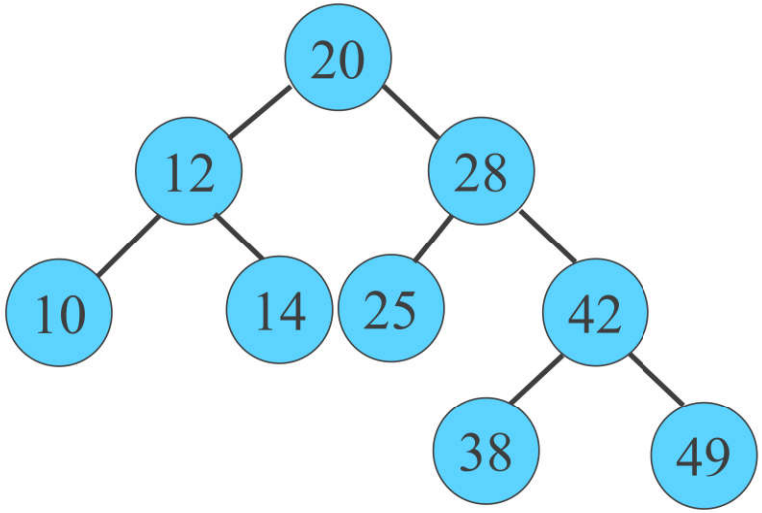
Double Rotation

- **Solution:**
  - ✓Inserting 15



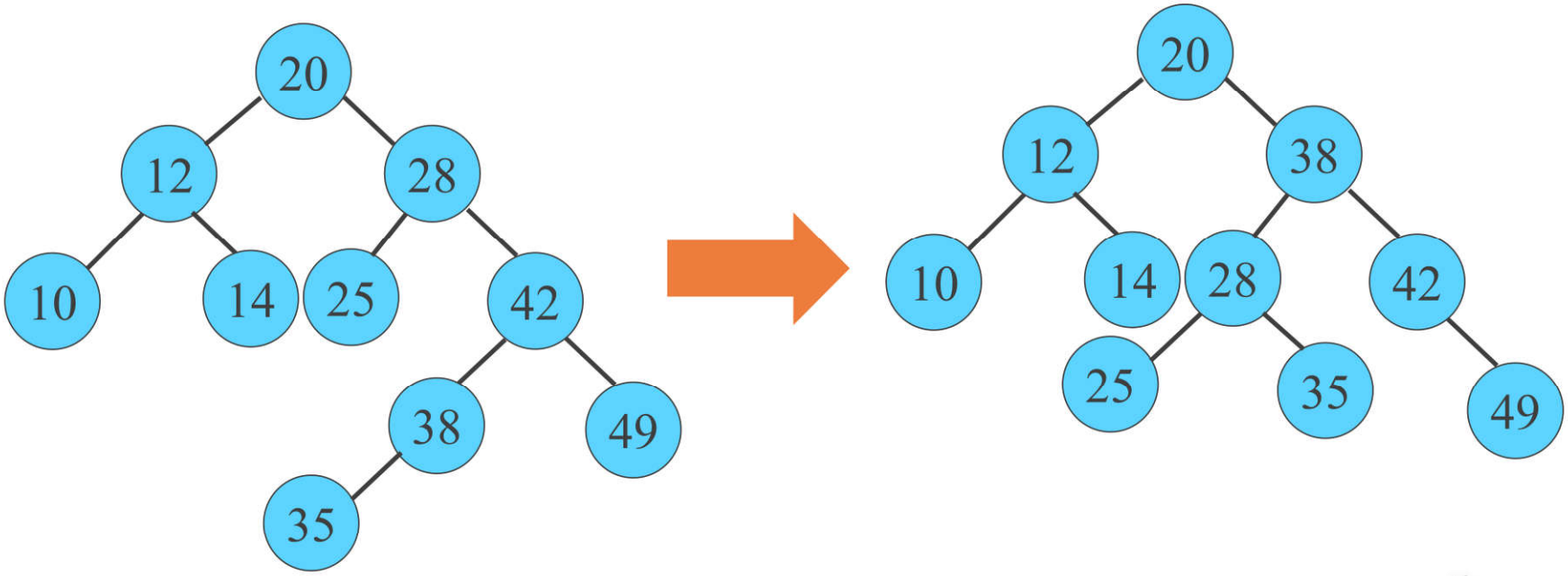
Double Rotation

- **Example:** Inserting 35 in the AVL tree



Double Rotation

- **Solution:**



Insertion

- Algorithm:
  - ✓ insert a node X in the AVL tree using the Binary search tree insertion
  - ✓ adjusts the balancing factor of all nodes in the path from the parent node of X to the root node
  - ✓ If the balancing factor of all nodes is +1 or 0 or -1

    then end the operation

    else adjust the balancing factor

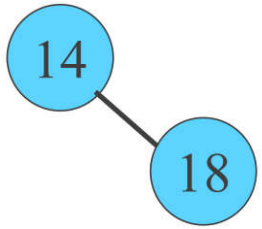
Insertion

- **Example:** Inserting 14, 18, 12, 9, 55, 5 and 13 in an empty AVL tree

Insertion

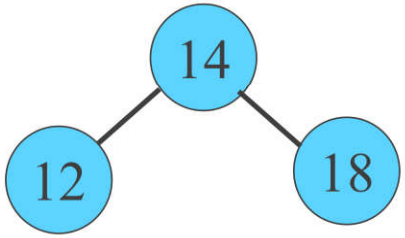
- Solution:
  - ✓inserting 14

  - ✓inserting 18

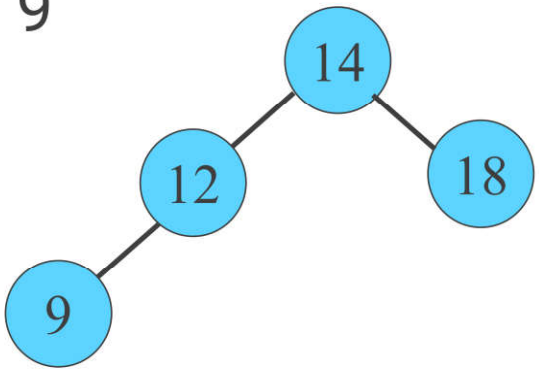


Insertion

- Solution:
  - ✓inserting 12

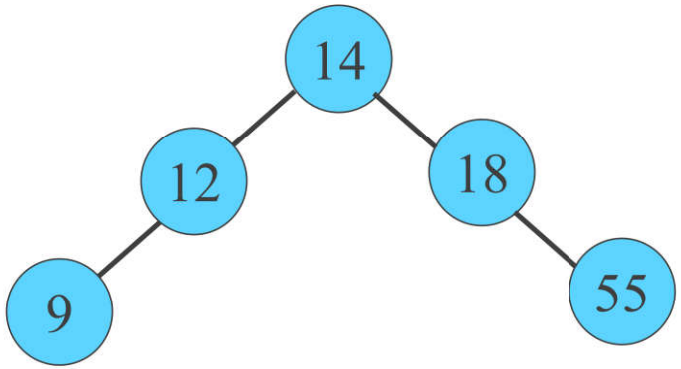


  - ✓inserting 9



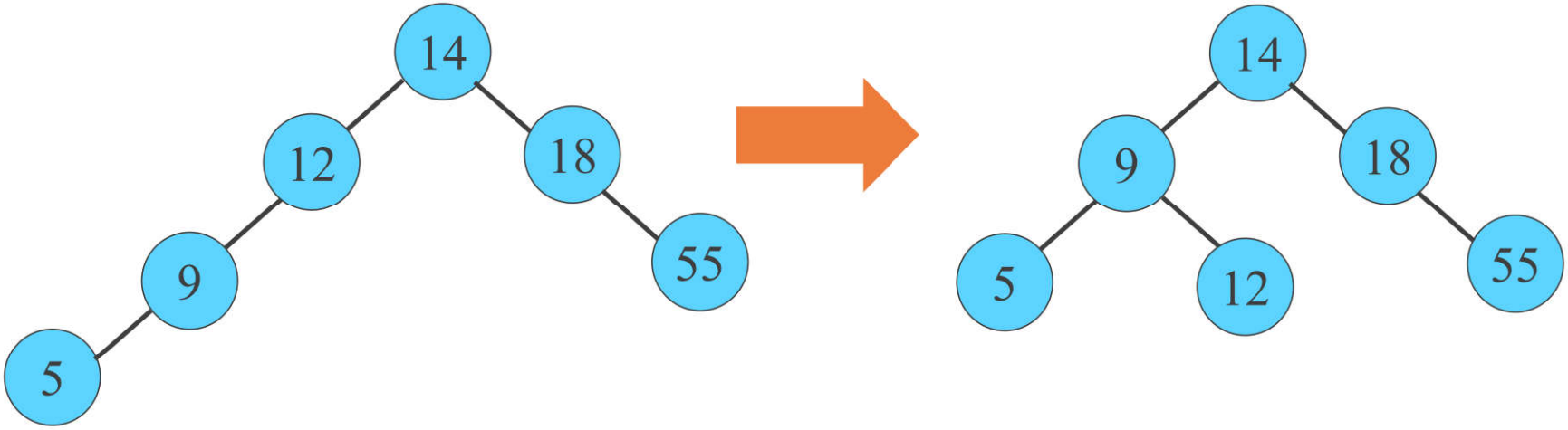
Insertion

- Solution:
  - ✓inserting 55



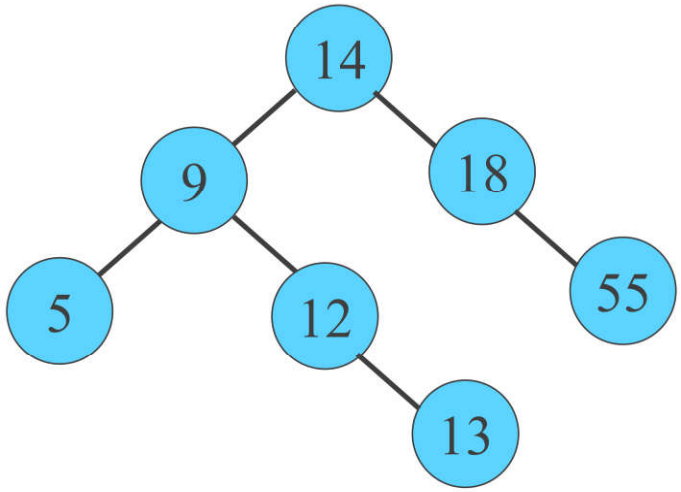
Insertion

- Solution:
  - ✓inserting 5



Insertion

- Solution:
  - ✓inserting 13



Deletion

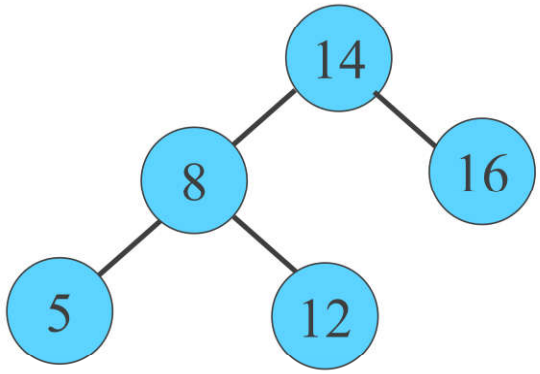
- Deletion is more complicated
- May need more than one rebalance on the path from deleted node to root
- Deletion of a node X from an AVL tree requires the same basic ideas, including single and double rotations, that are used for insertion

Deletion

- Algorithm:
  - ✓ delete a node X in the AVL tree using the Binary search tree deletion
  - ✓ adjusts the balancing factor of all nodes in the path from the parent node of X to the root node
  - ✓ If the balancing factor of all nodes is +1 or 0 or -1, end the operation
  - ✓ otherwise adjust the balancing factor

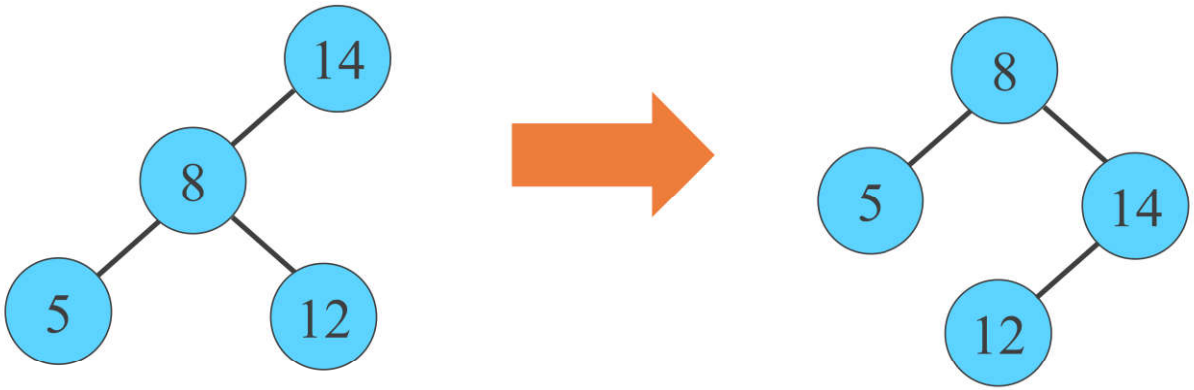
Deletion

- **Example:** Deleting 16 from the AVL tree



Deletion

- Solution:



Class Activity

Q & A

Formative Assessment