DATA STRUCTURES AND ITS APPLICATIONS

Balanced Trees

Sandesh B. J
Department of Computer Science & Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Balanced Trees

Sandesh B. J
Department of Computer Science & Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Balanced Trees

In this lecture you will be able to learn:

- **Why trees becomes unbalanced?**
- **Why we need to balance the tree?**
- **AVL Tree**
- **How do we balance the unbalanced trees using tree rotation techniques**
- **Different tree Rotation techniques**
  - ✓ **Left rotation, right rotation**
  - ✓ **Left-right rotation and right-left rotations**

Balanced Trees

Why Balanced Trees?

- Binary Search Tree(BST) – Data Structure used to implement the dictionary.

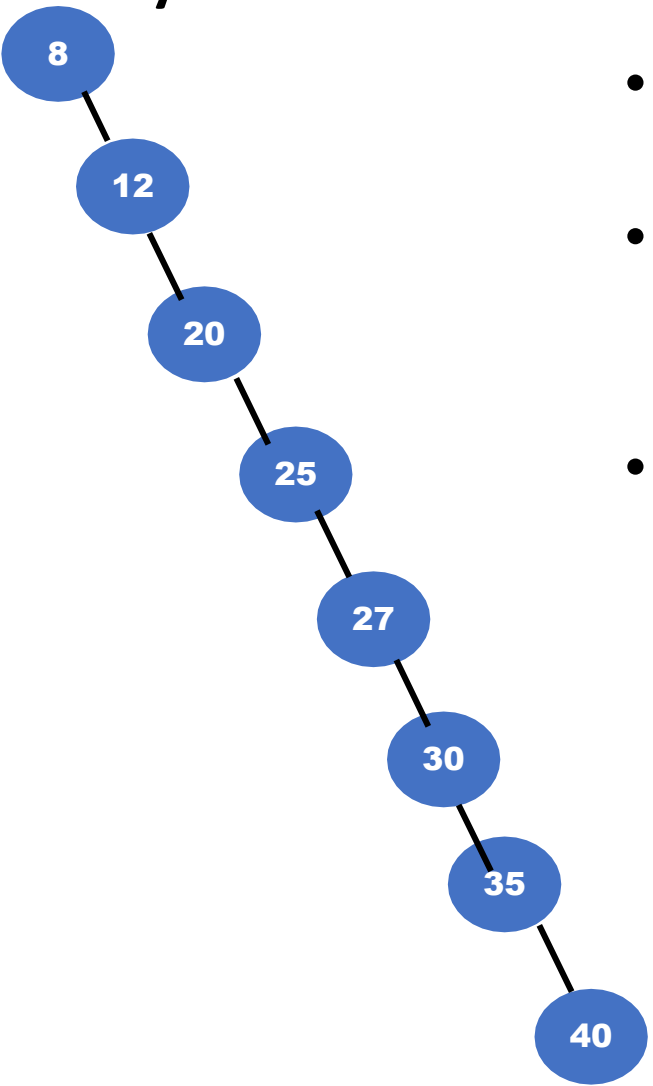
- What do we gain by implementing dictionary using BST instead of array ?

Complexity of Binary Search Tree

Operations	Average	Worst
Insert	O(log(n))	O(n)
Delete	O(log(n))	O(n)
Search	O(log(n))	O(n)

Balanced Trees

Why Balanced trees?



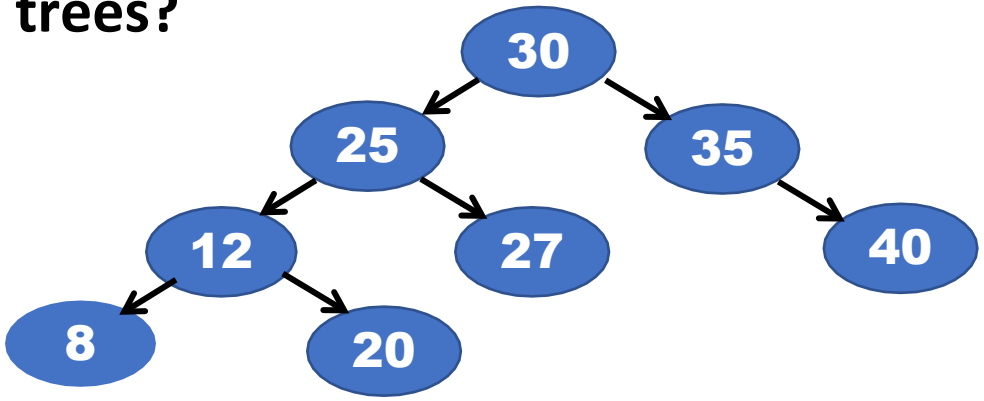
- Height of the BST depends on the order of insertion of elements into tree
- What happens if we insert elements given in increasing order?
  - Insert 8, 12, 20, 25, 27, 30, 35, 40
- Severely unbalanced

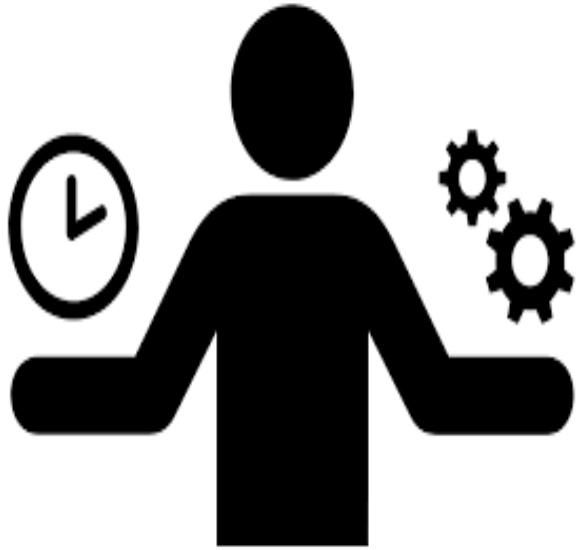
Disadvantages of Binary search trees

- The search and insertion algorithm does not ensure that the tree remains balanced
- The degree of balance dependent on the order in which the keys are inserted
- Tree can attain a height which can be as large as n-1
- Time taken for most of the operations worst case O(n)

Balanced Trees

Why Balanced trees?





- we can construct the binary search tree in which the height of the tree is always log(n)

Complexity of Balanced Binary Search Tree

Operations	Average	Worst
Insert	O(log(n))	O(log(n))
Delete	O(log(n))	O(log(n))
Search	O(log(n))	O(log()n)

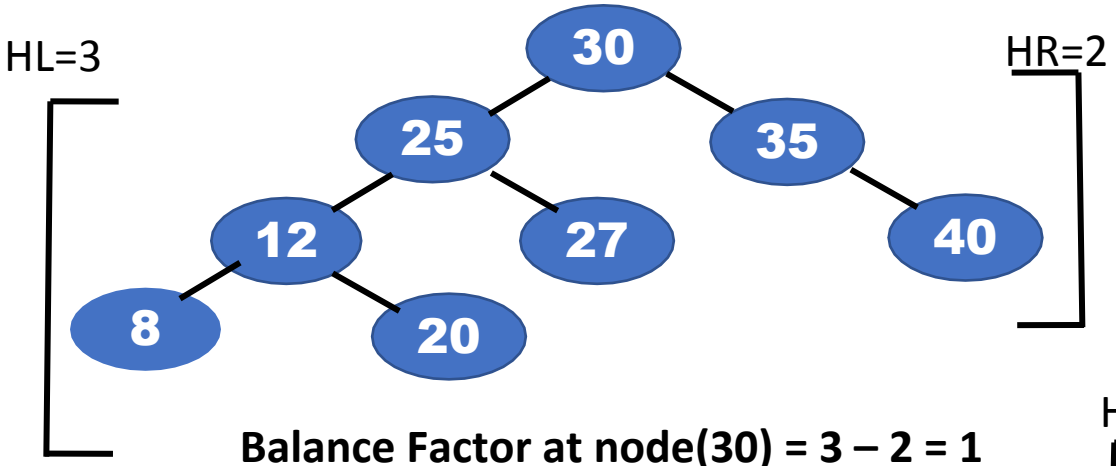
AVL Tree-Balanced Binary Search Trees

- AVL tree was invented in 1962 by two Russian mathematicians G.M. Adel'son-Vel'skii and E.M. Landis(AVL)

- An AVL tree is a binary search tree in which, for every node, the difference between the heights of the left and right subtrees, called the balance factor is either 0 or +1 or -1

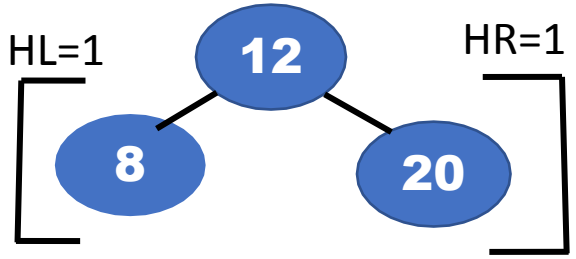
The Balance factor of any node:
Balance Factor = Height(left subtree) – Height(right subtree)

Balanced Binary Search Trees(AVL)

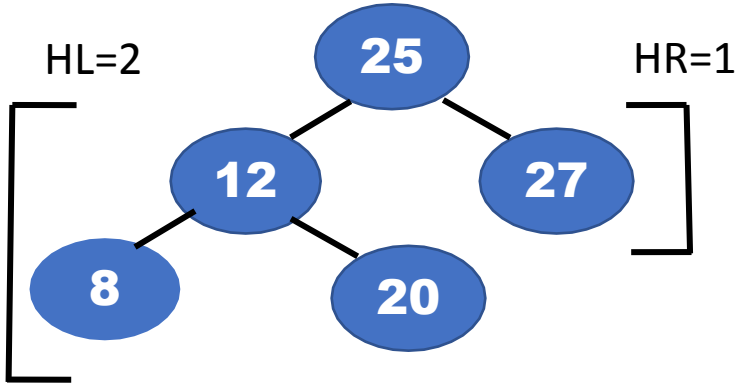


Balance Factor = HL - HR

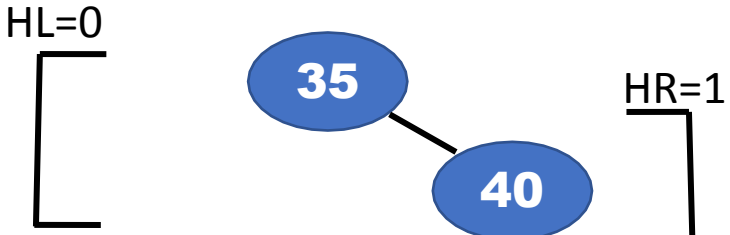
Balance Factor at node(30) = 3 – 2 = 1



Balance Factor at node(12) = 1– 1 = 0

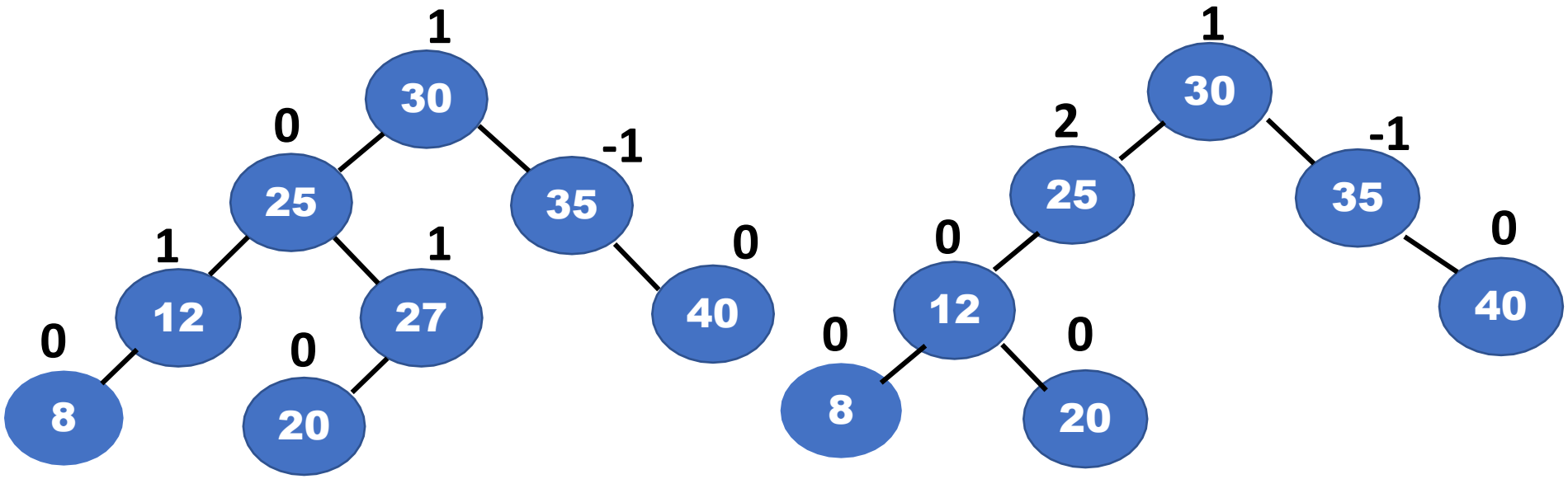


Balance Factor at node(25) = 2 – 1 = 1



Balance Factor at node(35) = 0 – 1 = -1

Balanced Binary Search Trees(AVL)



AVL Tree

Not an AVL tree

- Node in balanced binary tree has balance of
  1 – Height(left subtree) > Height(right subtree)
  0 – Height(left subtree) = Height(right subtree)
  -1 – Height(left subtree) < Height(right subtree)

Rotations – AVL Tree

- The AVL tree may become unbalanced after insert and delete operations

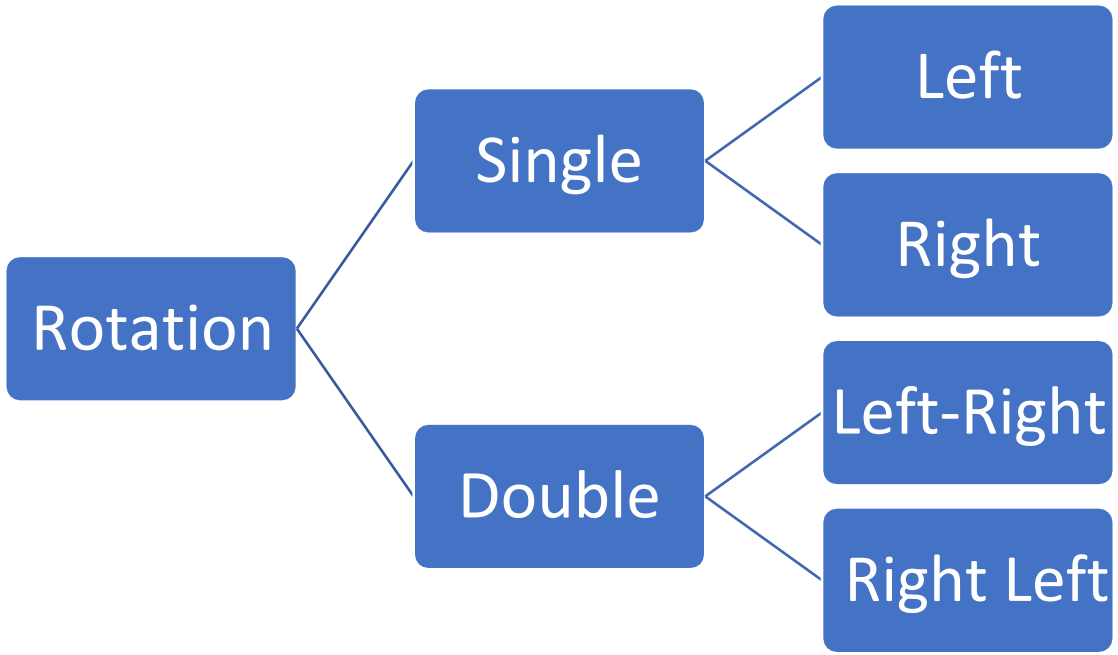
- If a key insertion violates the balance requirement at some node, the subtree rooted at that node is transformed via one of the four *rotations.*

- **Rotation** in a AVL tree is a local transformation of its subtree rooted at a node whose balance has become either +2 or -2

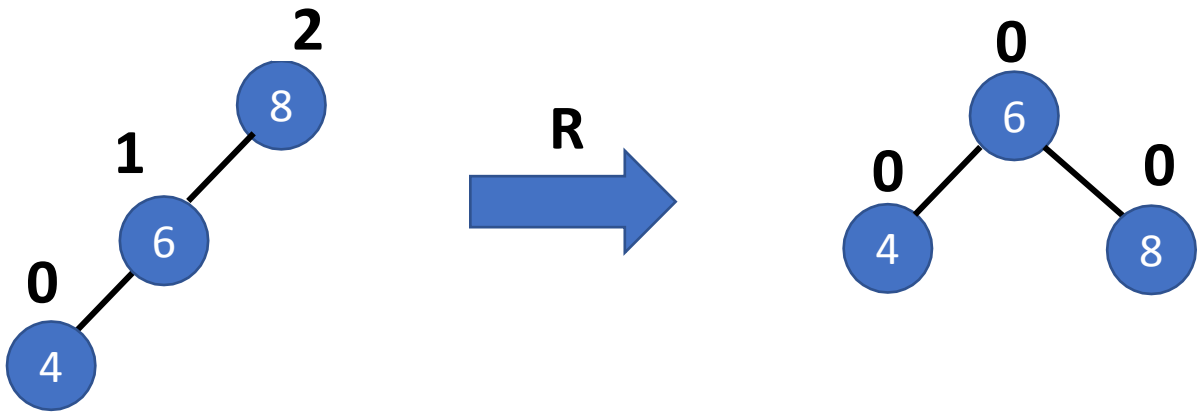
- In case there are several such nodes, The rotation is always performed for a subtree rooted at "unbalanced" node closest to the newly inserted leaf node.

Rotations – AVL Tree

- **Different types of Rotation:**



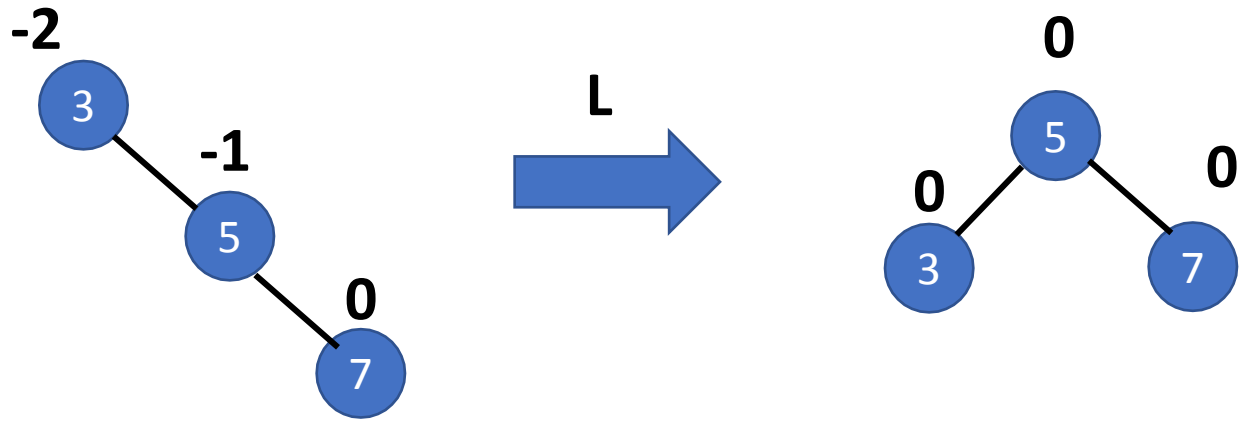
Single Right Rotation (R-Rotation)



- Root of the tree has balance of +1 before the insertion
- New key is inserted to the left of left child (Left-Left-case)
- Right rotation is performed at the subtree of the unbalanced node

Courtesy: “Introduction to Design and Analysis of Algorithms” By Anany Levitin

Single Left Rotation:

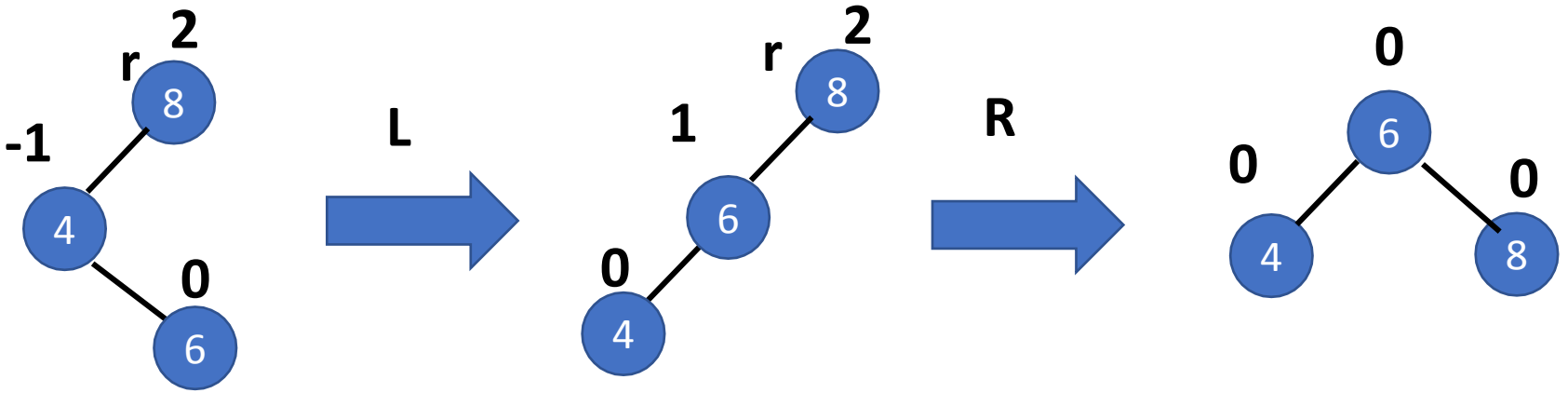


- Root of the tree had balance of -1 before the insertion
- New key is inserted to the right of right child (Right-Right-case)
- Left rotation is performed at the sub tree of the unbalanced node

Courtesy: “Introduction to Design and Analysis of Algorithms” By Anany Levitin

Rotations - AVL Trees

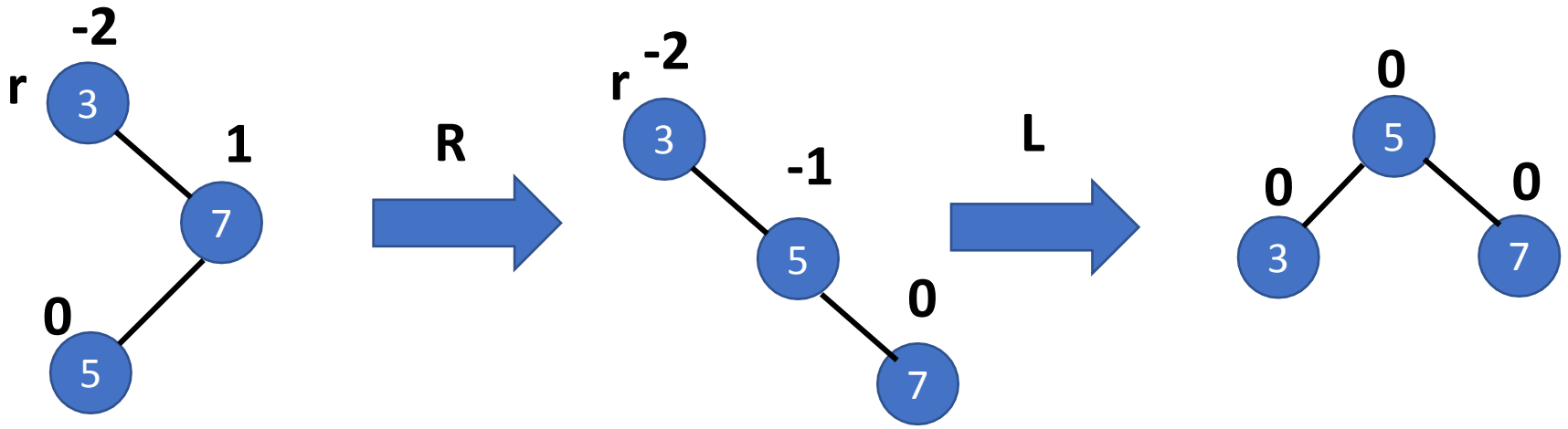
Double Left Right Rotation:



- Root r had the balance of +1 before the insertion
- New key is inserted to the right of left child (Left-Right-case)
- We perform the left rotation of left subtree of root r
- Right rotation of the new tree rooted at r

Courtesy: “Introduction to Design and Analysis of Algorithms” By Anany Levitin

Double Right Left Rotation:

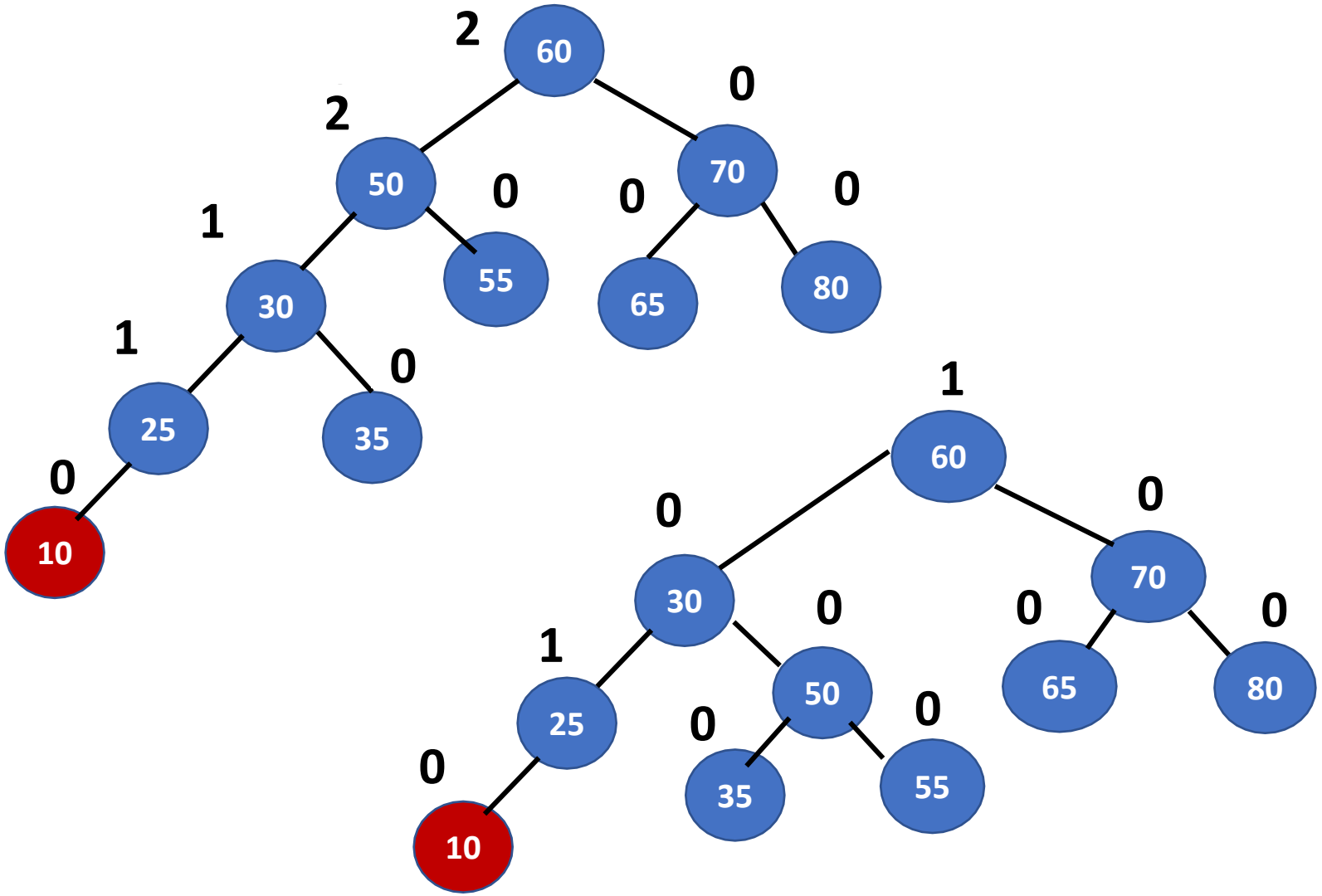


- Root r had the balance of -1 before the insertion
- New key is inserted to the left of right child (Right-Left-case)
- We perform the right rotation of right subtree of root r
- Left rotation of the new tree rooted at r

Courtesy: “Introduction to Design and Analysis of Algorithms” By Anany Levitin

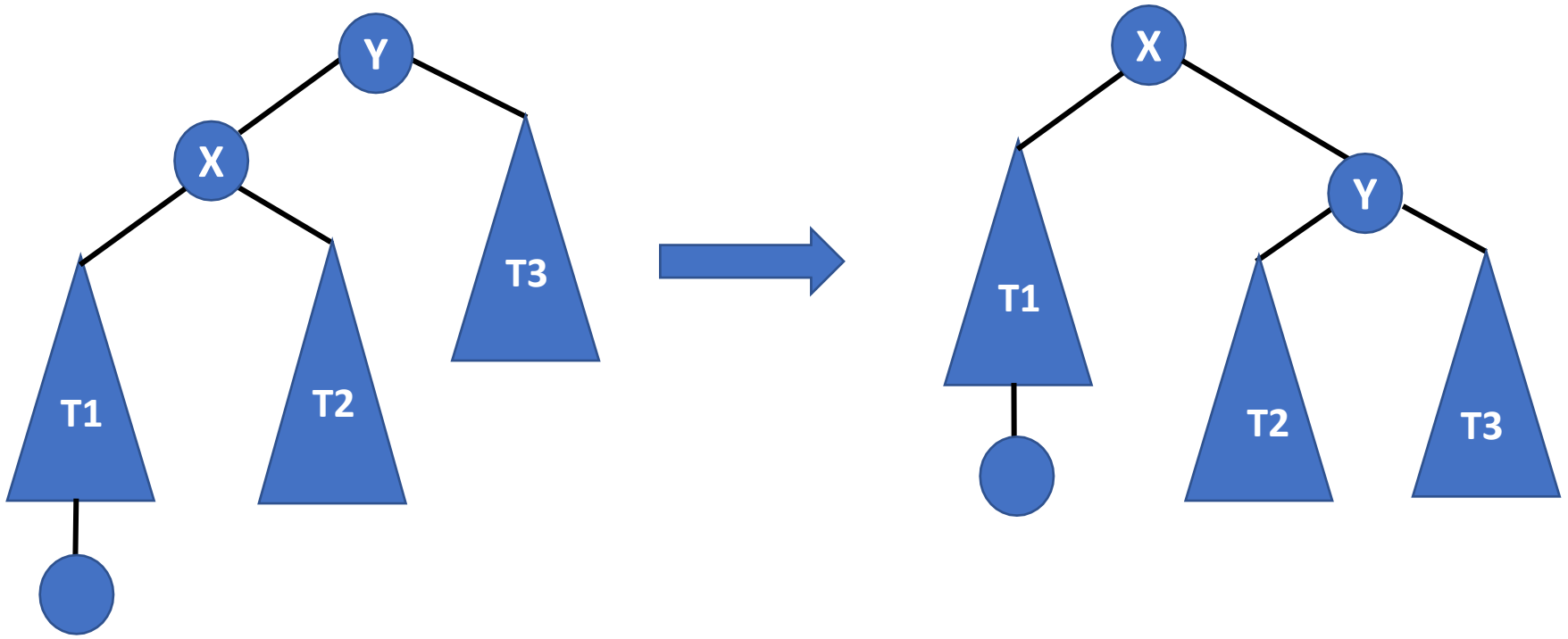
DATA STRUCTURES AND ITS APPLICATIONS

Example – Rotations in AVL Trees



Courtesy: “Introduction to Design and Analysis of Algorithms” By Anany Levitin

General form of Right-Rotation

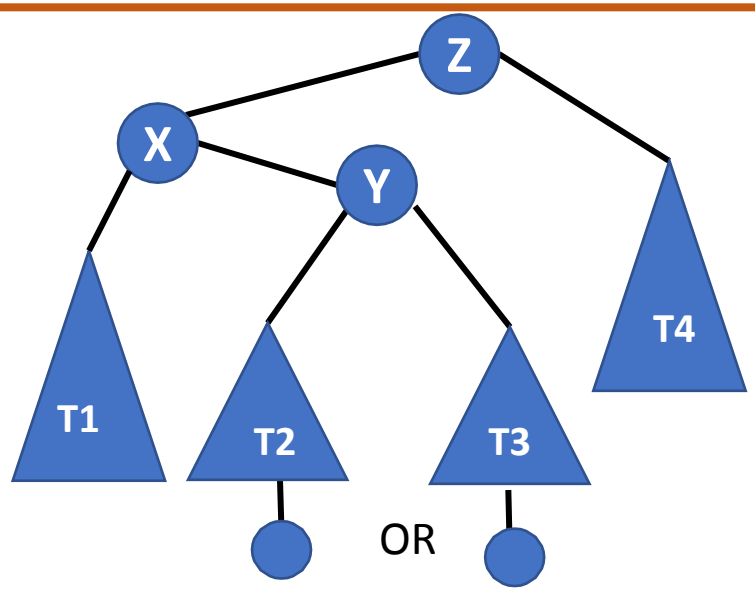


- keys(T1) < X < keys(T2) < Y < keys(T3)

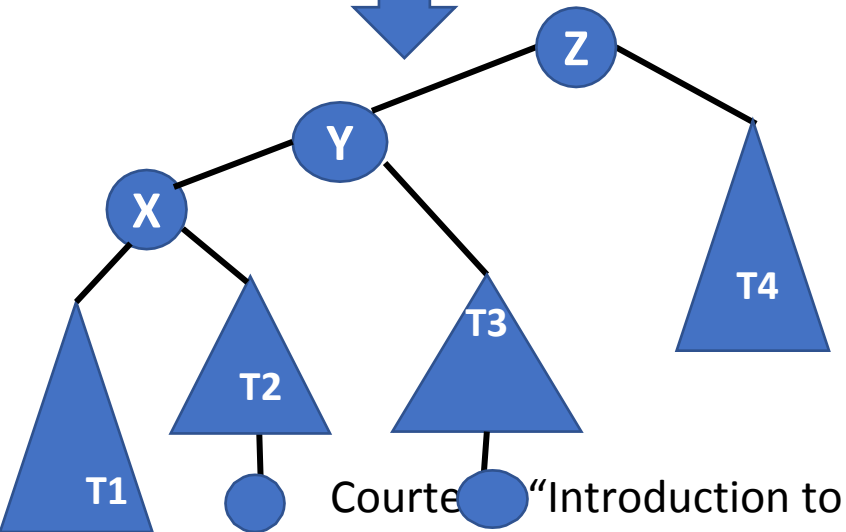
Courtesy: “Introduction to Design and Analysis of Algorithms” By Anany Levitin

DATA STRUCTURES AND ITS APPLICATIONS

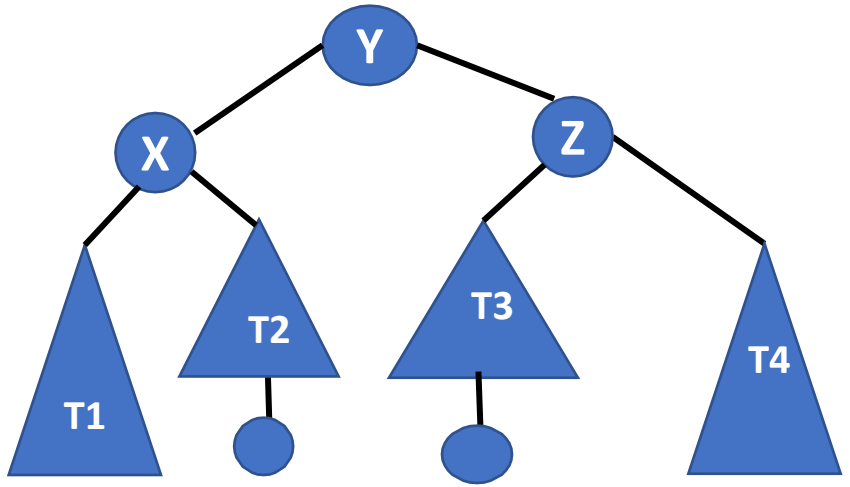
General form of Left – Right Rotation



OR



- keys(T1) < X <keys(T2) < Y < keys(T3) < Z < keys(T4)



Courte “Introduction to Design and Analysis of Algorithms” By Anany Levitin

THANK YOU

Sandesh B. J
Department of Computer Science & Engineering

sandesh_bj@pes.edu

+91 80 6618 6649