Advanced Algorithms & Data Structures

AIM OF THE SESSION

To familiarize students with the basic concept of AVL Tree

INSTRUCTIONAL OBJECTIVES

This Session is designed to:

1. Demonstrate the AVL Trees
2. List out the operations possible on the AVL Trees.
3. Describe the each operation
4. Describe the rotations of AVL Trees.

LEARNING OUTCOMES

At the end of this session, you should be able to:

1. Define AVL Trees
2. Describe four rotations of AVL Trees and their operations
3. Summarize definition, types and operations of AVL Trees and its applications

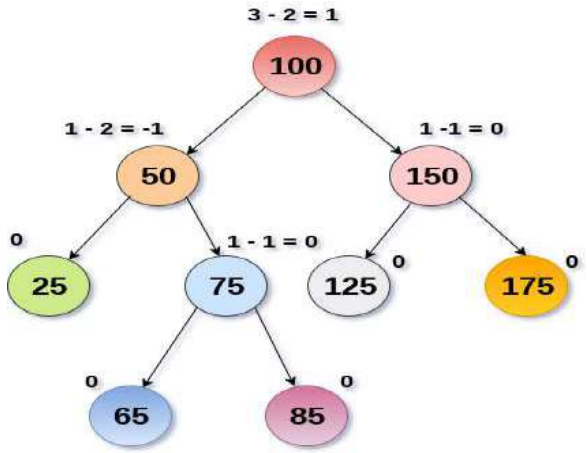
AVL Tree Introduction

AVL Tree is invented by GM Adelson - Velsky and EM Landis in 1962. The tree is named AVL in honour of its inventors.

AVL Tree can be defined as height balanced binary search tree in which each node is associated with a balance factor which is calculated by subtracting the height of its right sub-tree from that of its left sub-tree.

Tree is said to be balanced if balance factor of each node is in between -1 to 1, otherwise, the tree will be unbalanced and need to be balanced.

Balance Factor (k) = height (left(k)) - height (right(k))
If balance factor of any node is 1, it means that the left sub-tree is one level higher than the right sub-tree.
If balance factor of any node is 0, it means that the left sub-tree and right sub-tree contain equal height.
If balance factor of any node is -1, it means that the left sub-tree is one level lower than the right sub-tree.
An AVL tree is given in the following figure. We can see that, balance factor associated with each node is in between -1 and +1. therefore, it is an example of AVL tree.



AVL Tree

Complexity

Algorithm	Average case	Worst case
Space	o(n)	o(n)
Search	o(log n)	o(log n)
Insert	o(log n)	o(log n)
Delete	o(log n)	o(log n)

Operations on AVL tree

Due to the fact that, AVL tree is also a binary search tree therefore, all the operations are performed in the same way as they are performed in a binary search tree. Searching and traversing do not lead to the violation in property of AVL tree.

SN	Operation	Description
1	Insertion	Insertion in AVL tree is performed in the same way as it is performed in a binary search tree. However, it may lead to violation in the AVL tree property and therefore the tree may need balancing. The tree can be balanced by applying rotations.
2	Deletion	Deletion can also be performed in the same way as it is performed in a binary search tree. Deletion may also disturb the balance of the tree therefore, various types of rotations are used to rebalance the tree.

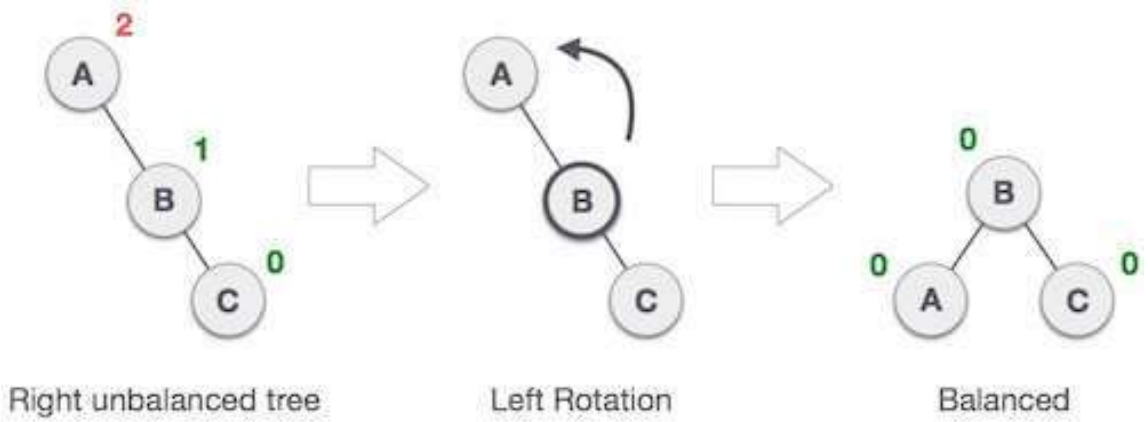
AVL Rotations

We perform rotation in AVL tree only in case if Balance Factor is other than **-1, 0, and 1**. There are basically four types of rotations which are as follows:

1.L L rotation: Inserted node is in the left subtree of left subtree of A
2.R R rotation : Inserted node is in the right subtree of right subtree of A
3.L R rotation : Inserted node is in the right subtree of left subtree of A
4.R L rotation : Inserted node is in the left subtree of right subtree of A

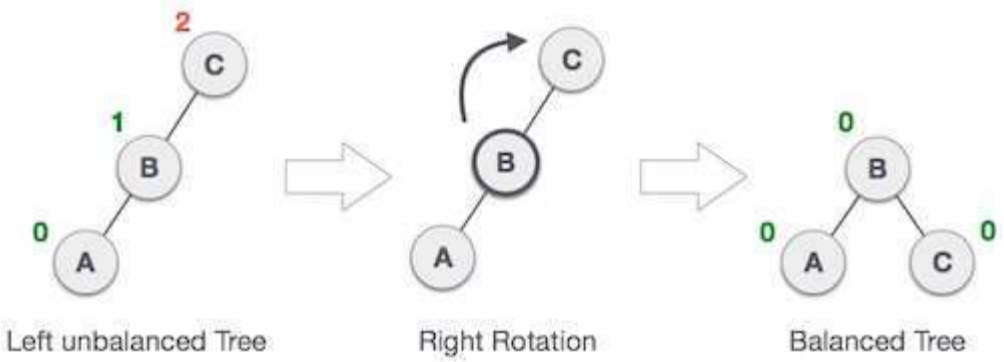
1. RR Rotation

When BST becomes unbalanced, due to a node is inserted into the right subtree of the right subtree of A, then we perform RR rotation, RR rotation is an anticlockwise rotation, which is applied on the edge below a node having balance factor -2



2. LL Rotation

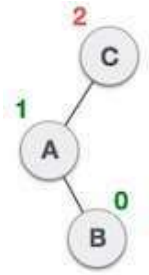
When BST becomes unbalanced, due to a node is inserted into the left subtree of the left subtree of C, then we perform LL rotation, LL rotation is clockwise rotation, which is applied on the edge below a node having balance factor 2.



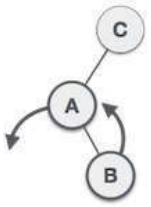
Above example, node C has balance factor 2 because a node A is inserted in the left subtree of C left subtree. We perform the LL rotation on the edge below A.

3. LR Rotation

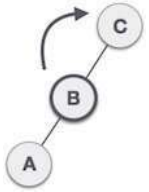
Double rotations are bit tougher than single rotation which has already explained above. LR rotation = RR rotation + LL rotation, i.e., first RR rotation is performed on subtree and then LL rotation is performed on full tree, by full tree we mean the first node from the path of inserted node whose balance factor is other than -1, 0, or 1.



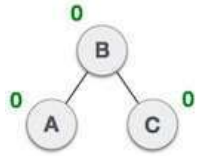
A node B has been inserted into the right subtree of A the left subtree of C, because of which C has become an unbalanced node having balance factor 2. This case is L R rotation where: Inserted node is in the right subtree of left subtree of C



After performing RR rotation, node C is still unbalanced, i.e., having balance factor 2, as inserted node A is in the left of left of **C**



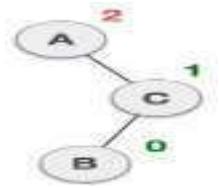
Now we perform LL clockwise rotation on full tree, i.e. on node C.
node **C** has now become the right subtree of node B, A is left subtree of B



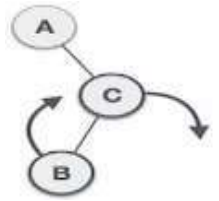
Balance factor of each node is now either -1, 0, or 1, i.e. BST is balanced now.

4. RL Rotation

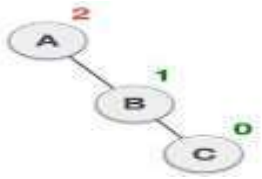
As already discussed, that double rotations are bit tougher than single rotation which has already explained above. R L rotation = LL rotation + RR rotation, i.e., first LL rotation is performed on subtree and then RR rotation is performed on full tree, by full tree we mean the first node from the path of inserted node whose balance factor is other than -1, 0, or 1.



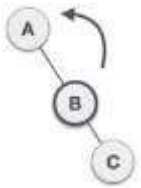
A node **B** has been inserted into the left subtree of **C** the right subtree of **A**, because of which A has become an unbalanced node having balance factor - 2. This case is RL rotation where: Inserted node is in the left subtree of right subtree of A



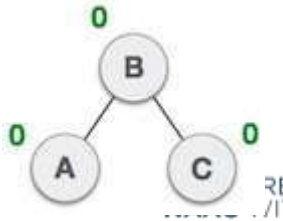
As RL rotation = LL rotation + RR rotation, hence, LL (clockwise) on subtree rooted at **C** is performed first. By doing RR rotation, node **C** has become the right subtree of **B**.



After performing LL rotation, node **A** is still unbalanced, i.e. having balance factor -2, which is because of the right-subtree of the right-subtree node A.



Now we perform RR rotation (anticlockwise rotation) on full tree, i.e. on node A. node **C** has now become the right subtree of node B, and node A has become the left subtree of B.



Balance factor of each node is now either -1, 0, or 1, i.e., BST is balanced now.

AVL TREES

AVL Tree:

- One of the more popular balanced trees, know as an AVL tree, was introduced in 1962 by Adelson-Velski and Landis.

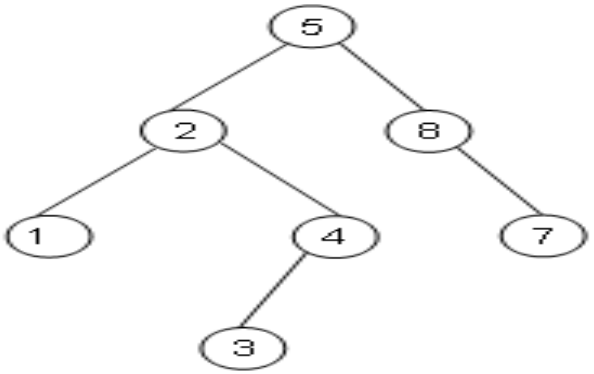
Definition: 1

- An empty binary tree is an AVL tree. If T is a nonempty binary tree with T_L and T_R as its left and right sub trees, the T is an AVL tree iff
- 1. T_L and T_R are AVL trees and
- 2. $|h_L - h_R| <= 1$ where h_L and h_R are the heights of T_L and T_R, respectively.
- An AVL search tree is a binary search tree that is also an AVL tree.

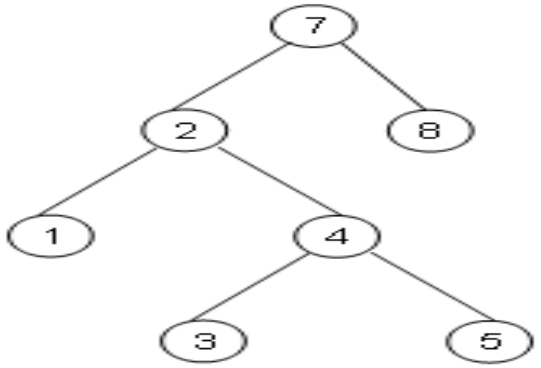
Definition: 2

- An AVL tree is a binary search tree in which for *every* node in the tree, the height of the left and right sub trees differ by at most 1.

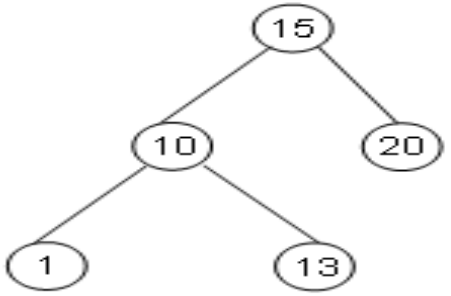
Examples:



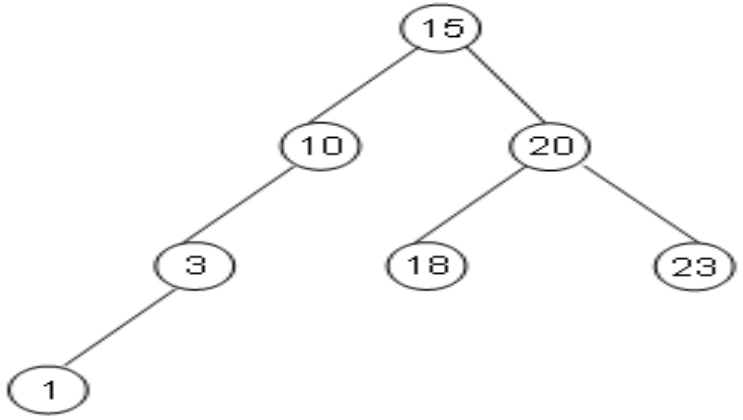
Fig(a) An AVL Tree



Fig(b) Not An Tree

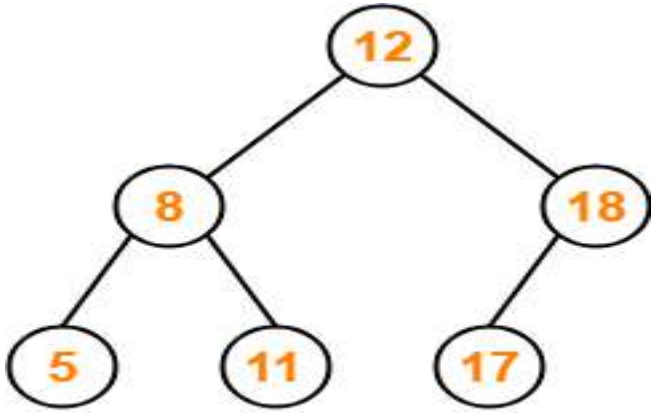


Fig(c) An AVL Tree

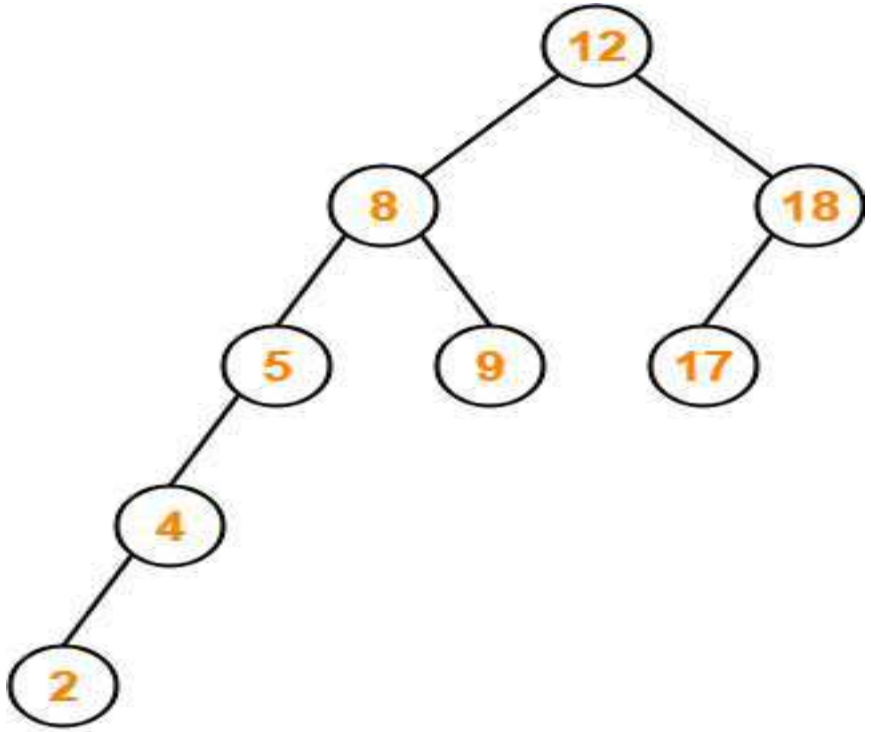


Fig(d) Not AVL Tree

Examples:



AVL Tree Example



Not an AVL Tree

Balance Factor:

- In AVL tree, Balance factor is defined for every node.
- Balance factor of a node = Height of its left subtree – Height of its right subtree
- In AVL tree, Balance factor of every node is either 0 or 1 or -1.

AVL Tree Operations:

- Like **BST Operations**, commonly performed operations on AVL tree are:
    a. Search Operation
    b. Insertion Operation
    c. Deletion Operation

- After performing any operation on AVL tree, the balance factor of each node is checked.
- There are following two cases possible:

Case-01:

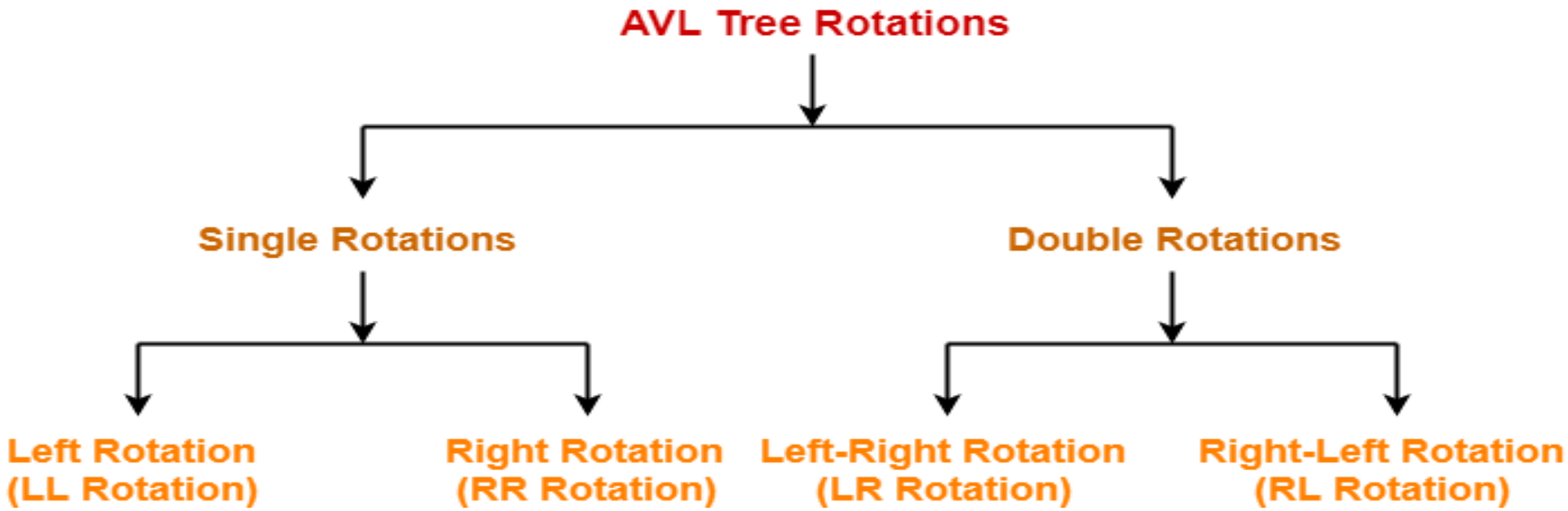
- After the operation, the balance factor of each node is either 0 or 1 or -1.
- In this case, the AVL tree is considered to be balanced.
- The operation is concluded.

Case-02:

- After the operation, the balance factor of at least one node is not 0 or 1 or -1.
- In this case, the AVL tree is considered to be imbalanced.
- Rotations are then performed to balance the tree.

AVL Tree Rotations:

- Rotation is the process of moving the nodes to make tree balanced.

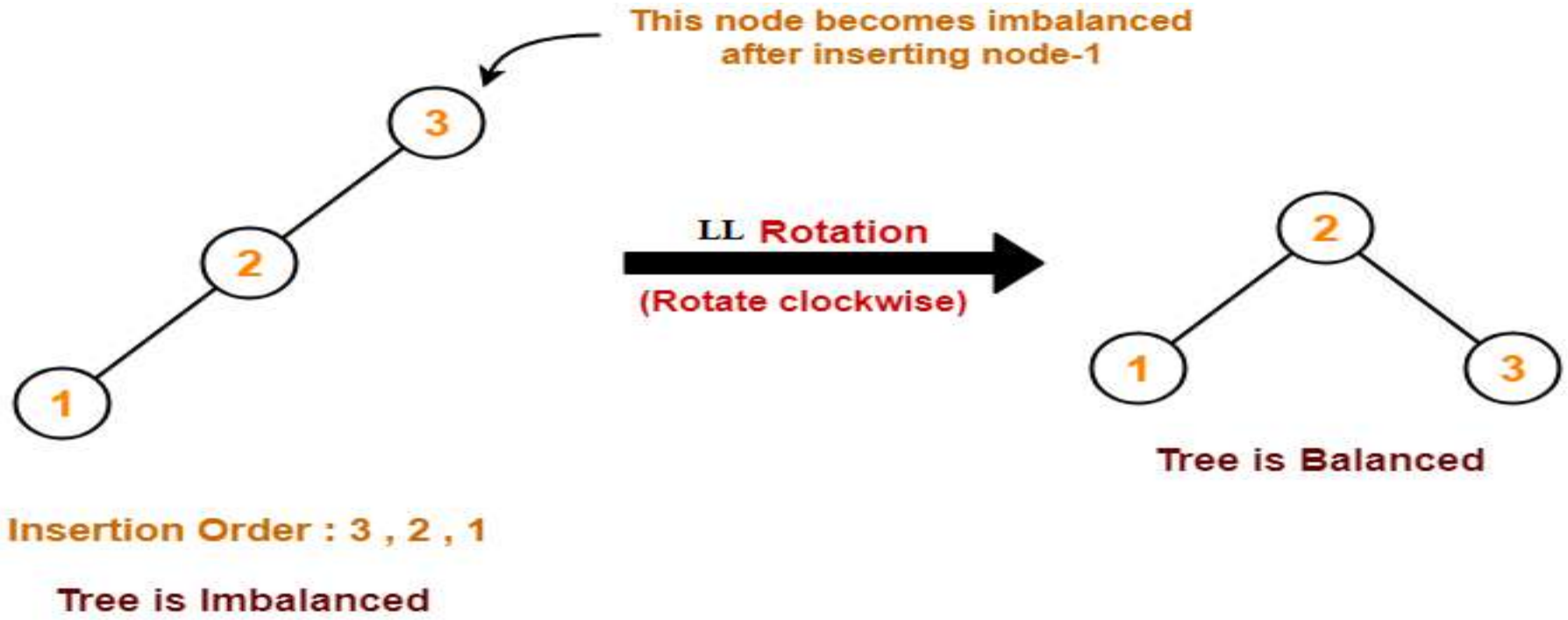


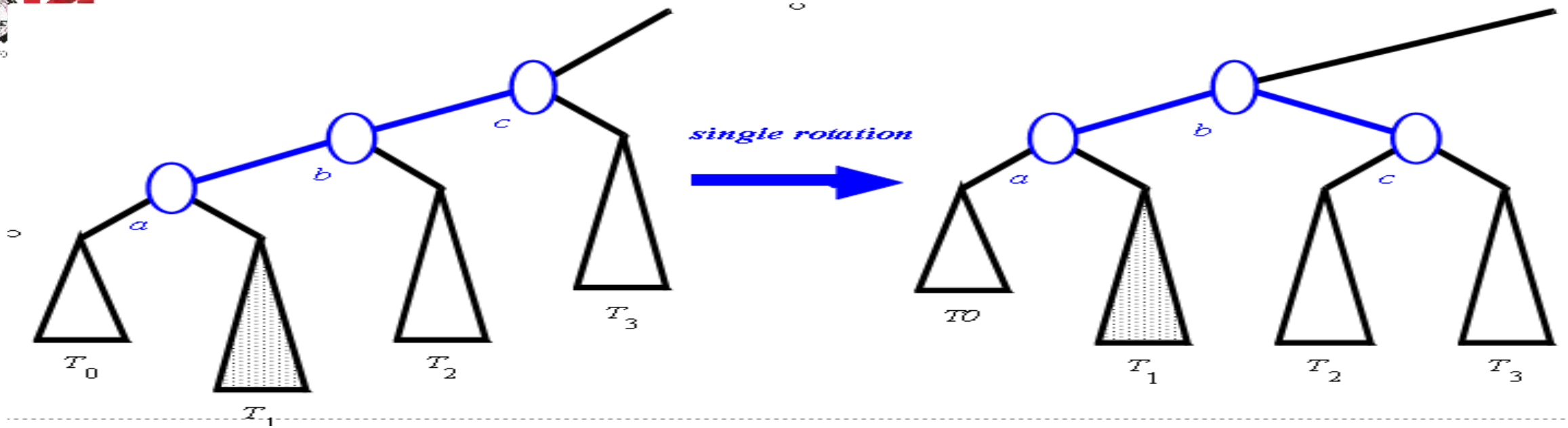
1. Left Rotation (LL Rotation)
2. Right Rotation (RR Rotation)
3. Left-Right Rotation (LR Rotation)
4. Right-Left Rotation (RL Rotation)

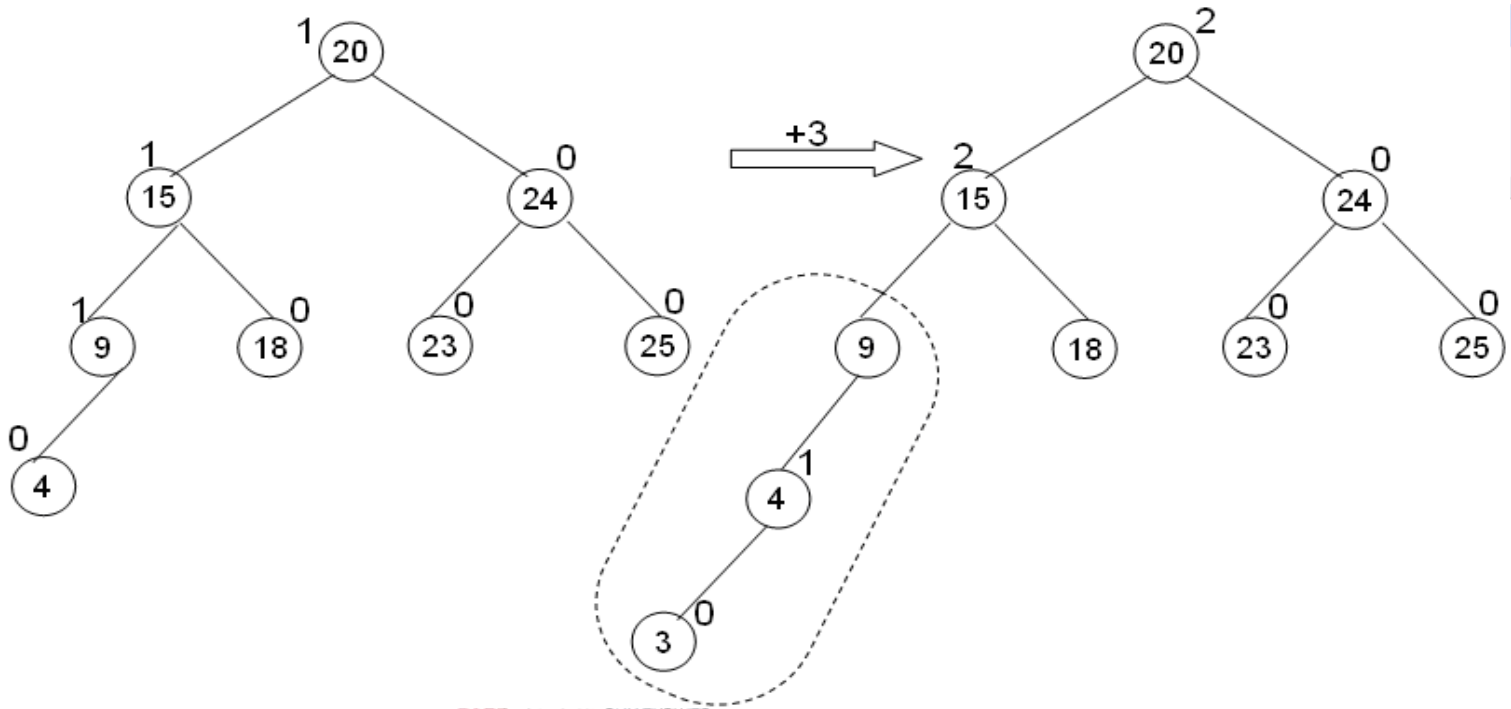
Rotations:

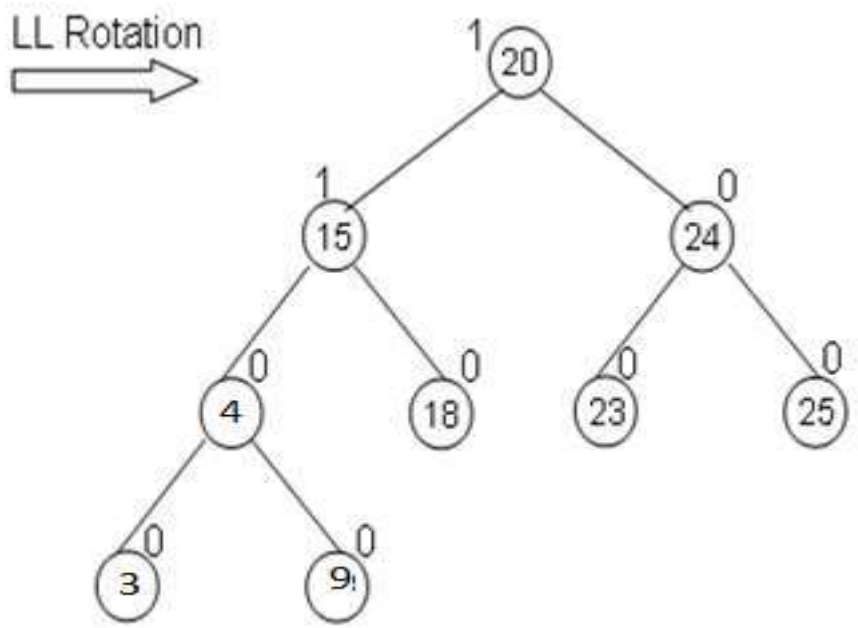
- The insert and delete operations of AVL tree are the same as binary search tree (BST).
- Since an insertion (deletion) involves adding (deleting) a tree node, this can only increase (decrease) the heights of some subtree(s) by 1.
- Thus, the AVL tree property may be violated.
- If the AVL tree property is violated at a node x, it means that the heights of left(x) and right(x) differ by exactly 2.
- After the insertion or deletion operations, we need to examine the tree and see if any node violates the AVL tree property.
- If the AVL tree property is violated at node *x*, single or double rotation will be applied to *x* to restore the AVL tree property.
- Rotation will be applied in a bottom-up manner starting at the place of insertion (deletion).
- Thus, when we perform a rotation at *x*, the AVL tree property is restored at all proper descendants of *x*. This fact is important.

LL Rotation:

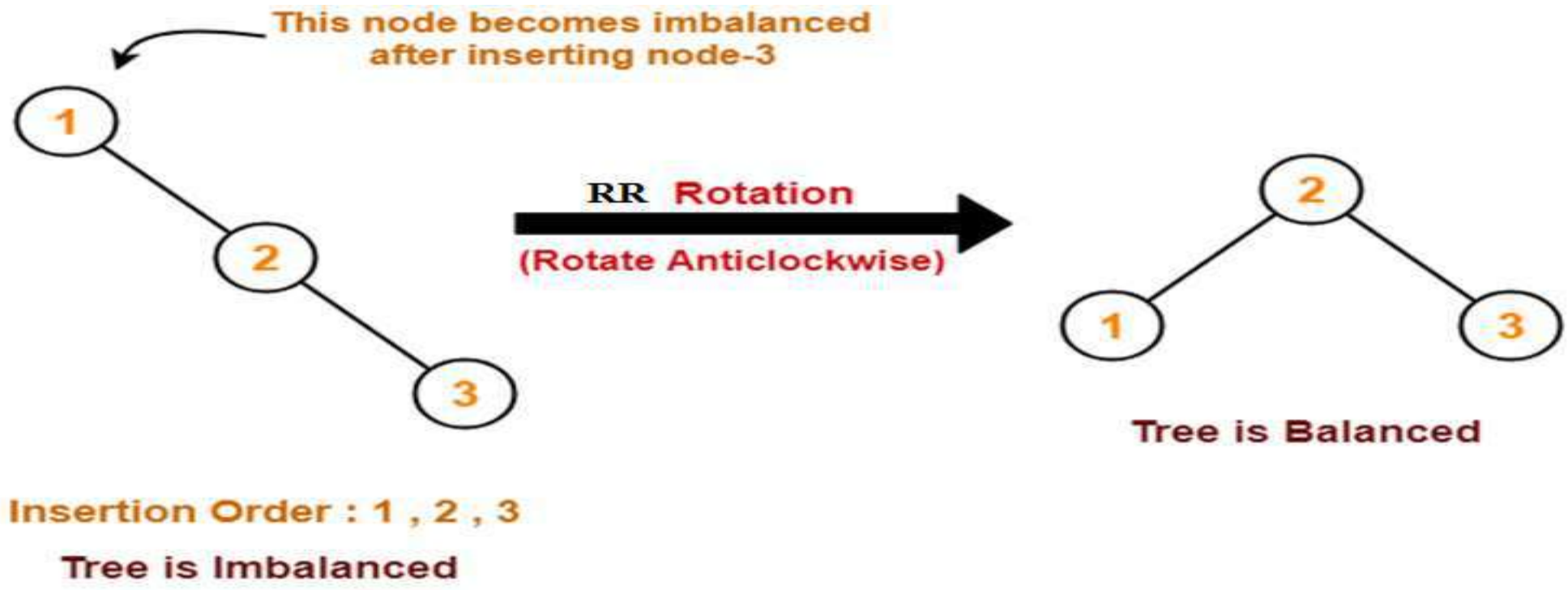

This node becomes imbalanced
after inserting node-1
3
2
1
LL Rotation
(Rotate clockwise)
2
1
3
Tree is Balanced
Insertion Order : 3 , 2 , 1
Tree is Imbalanced

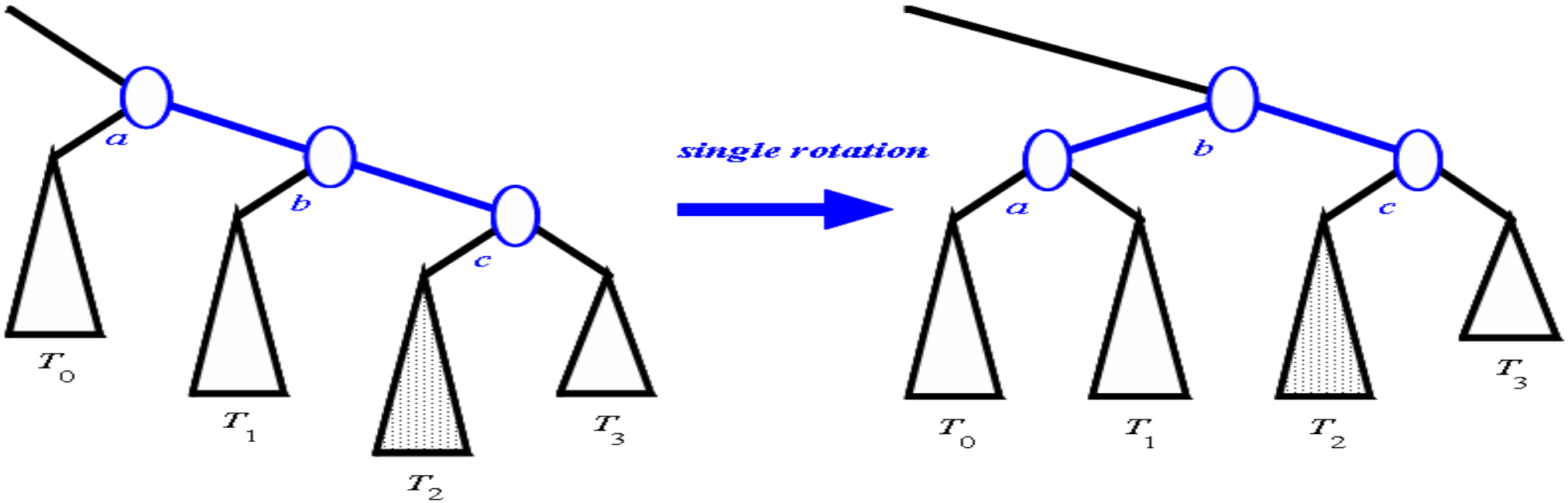

a
b
c
T0
T1
T2
T3
single rotation
a
b
c
T0
T1
T2
T3

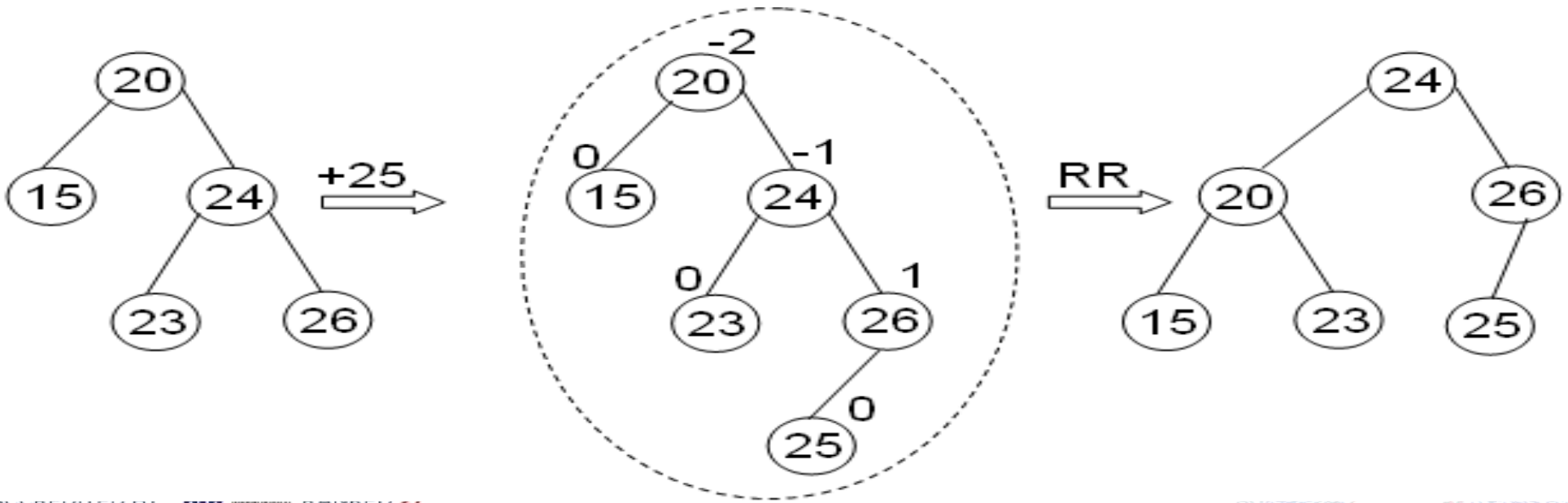

20
15
24
9
18
23
25
4
+3
3


LL Rotation
20
15
24
4
18
23
25
3

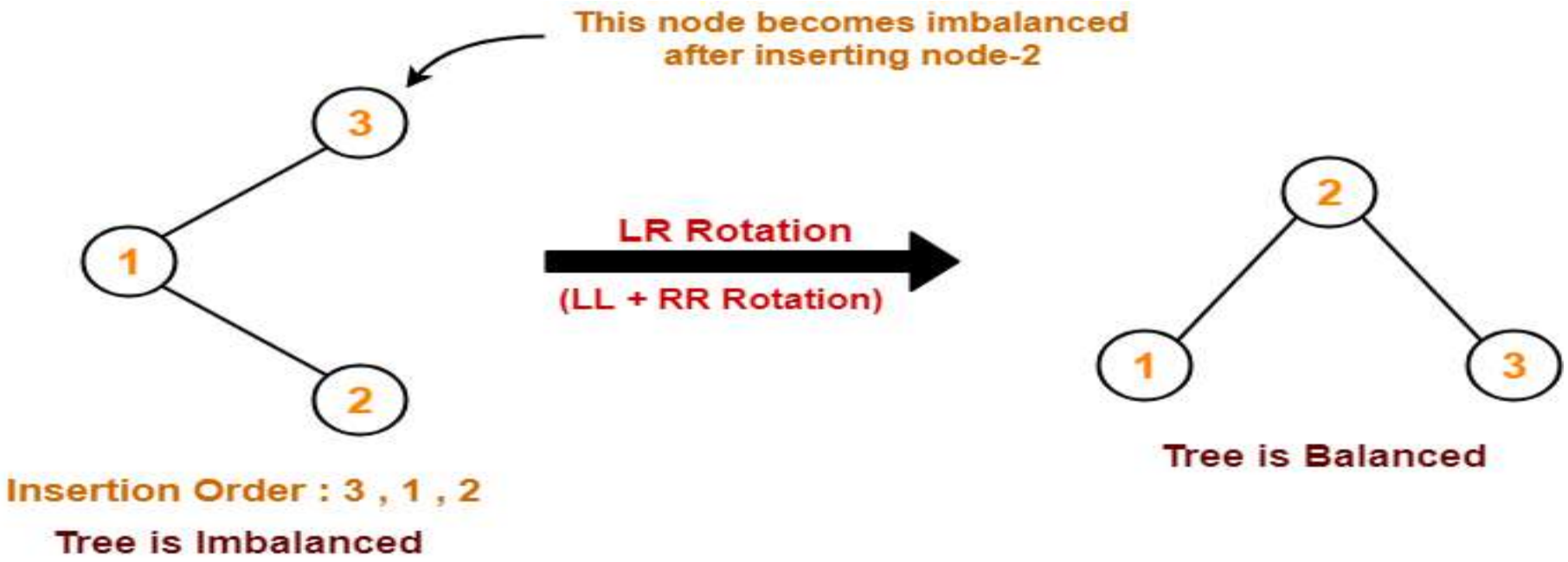
RR Rotation:

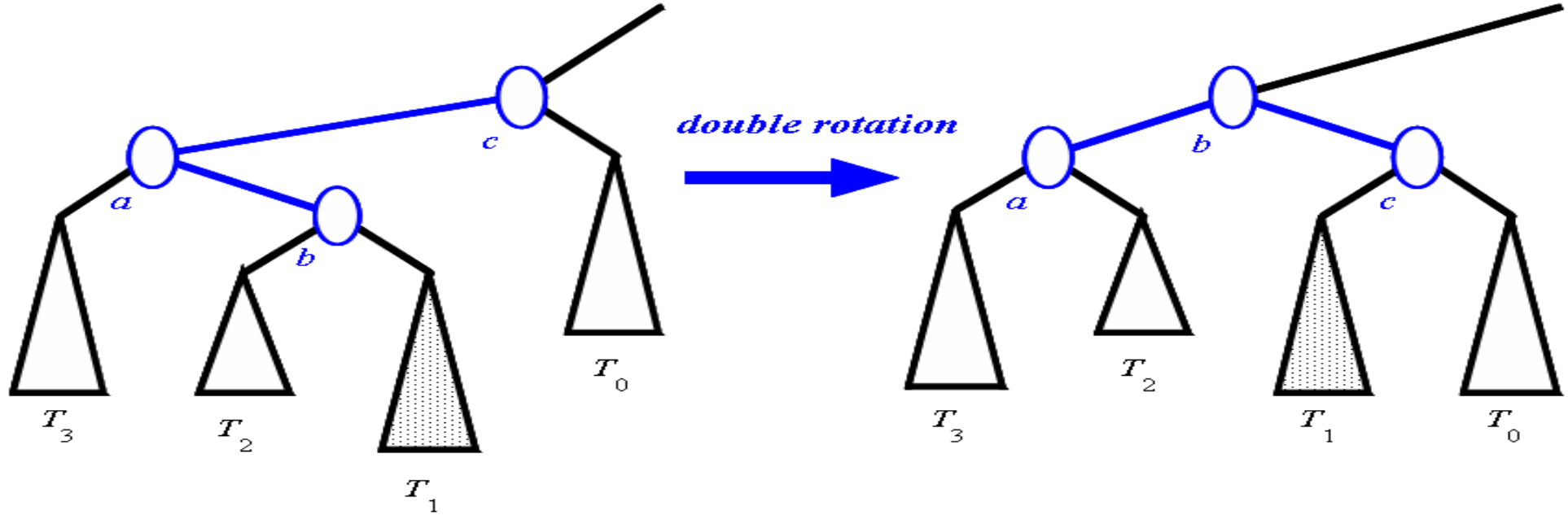

This node becomes imbalanced
after inserting node-3
1
2
3
RR Rotation
(Rotate Anticlockwise)
2
1
3
Tree is Balanced
Insertion Order : 1 , 2 , 3
Tree is Imbalanced

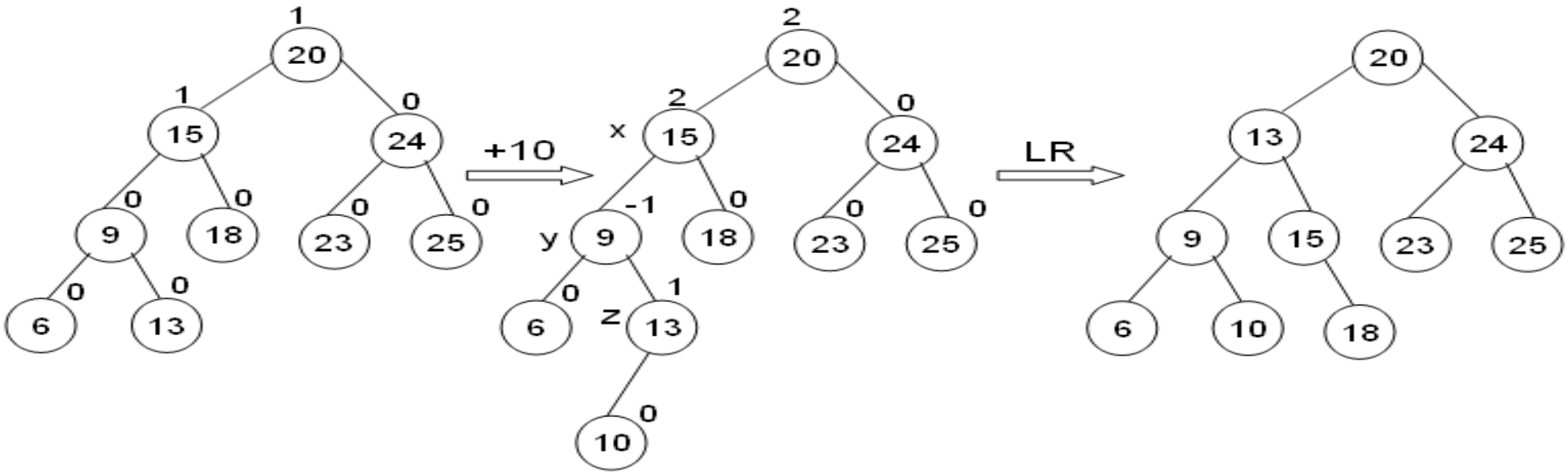




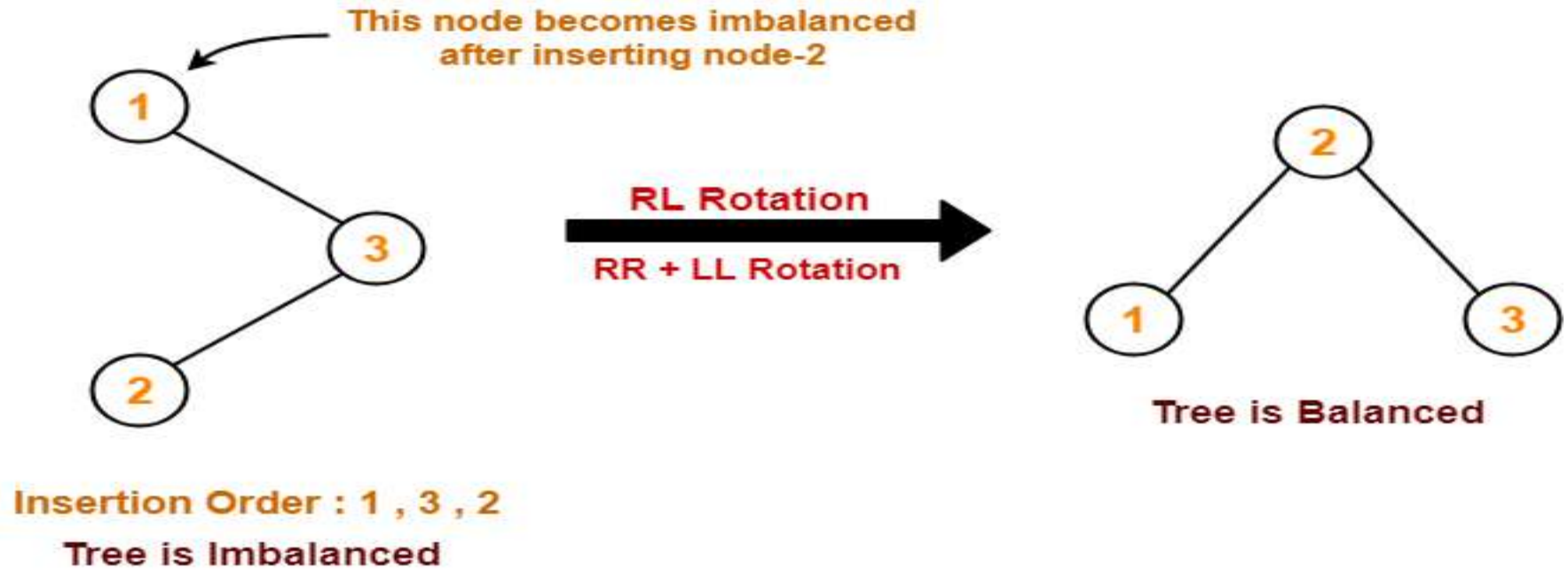
LR Rotation:

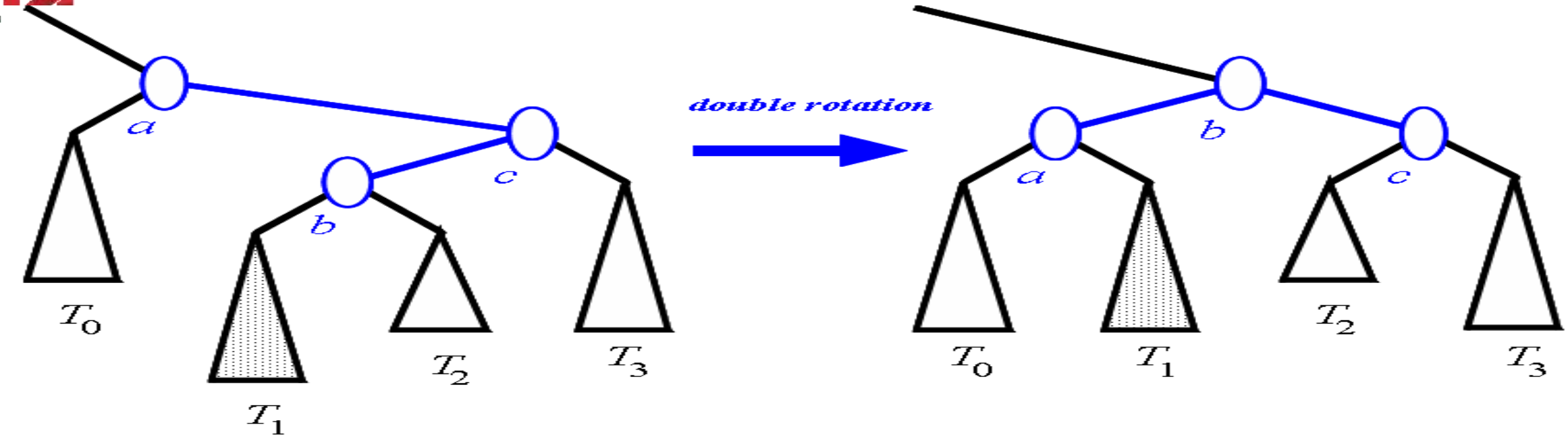

This node becomes imbalanced
after inserting node-2
3
1
2
LR Rotation
(LL + RR Rotation)
2
1
3
Insertion Order : 3 , 1 , 2
Tree is Imbalanced
Tree is Balanced

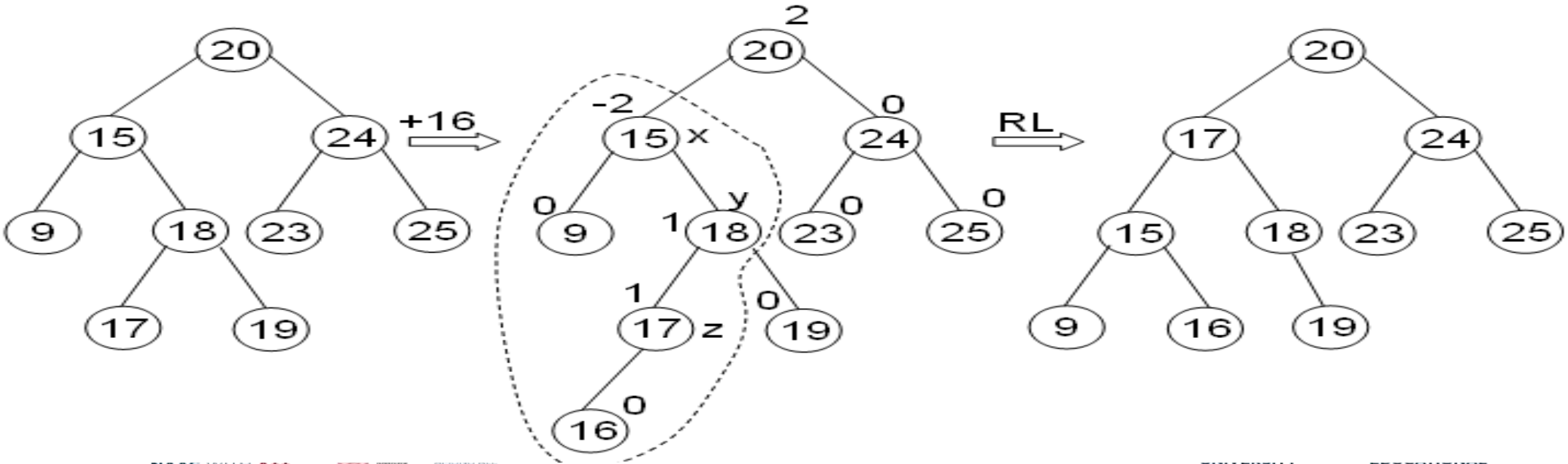

c
a
b
T_3
T_2
T_1
T_0
double rotation
b
a
c
T_3
T_2
T_1
T_0


1
20
1
15
0
24
0
9
0
18
0
23
0
25
0
6
0
13
+10
2
20
x
2
15
0
24
y
-1
9
0
18
0
23
0
25
0
6
z
1
13
0
10
LR
20
13
24
9
15
23
25
6
10
18

RL Rotation:


This node becomes imbalanced
after inserting node-2
1
3
2
RL Rotation
RR + LL Rotation
2
1
3
Tree is Balanced
Insertion Order : 1 , 3 , 2
Tree is Imbalanced


a
b
c
T_0
T_1
T_2
T_3
double rotation
b
a
c
T_0
T_1
T_2
T_3


20
15
24
9
18
23
25
17
19
+16
2
20
-2
0
15
x
24
0
y
9
1
18
23
0
25
0
1
0
17
z
19
1
0
16
RL
20
17
24
15
18
23
25
9
16
19

Insertion:

- Perform normal BST insertion.
- Check and restore AVL tree property.
  - Trace from path of inserted leaf towards the root, and check if the AVL tree property is violated.
  - Check to see if heights of left(x) and right(x) height differ by at most 1.
  - If the AVL tree property is violated, there are 4 rotation cases to restore the AVL tree property.
- Trace from path of inserted leaf towards the root, and check if the AVL tree property is violated. Perform rotation if necessary.

- For insertion, once we perform (single or double) rotation at a node *x*, the AVL tree property is already restored. We need not to perform any rotation at any ancestor of x.
- Thus one rotation is enough to restore the AVL tree property.
- There are 4 different cases (actually 2), so don't mix up them!
- The time complexity to perform a rotation is O(1).
- The time complexity to insert, and find a node that violates the AVL property is dependent on the height of the tree, which is O(log(n)).
- So insertion takes O(log(n)).

Deletion:

- Delete a node x as in ordinary BST (Note that x is either a leaf or x has exactly one child.).
- Check and restore the AVL tree property.
- Trace from path of deleted node towards the root, and check if the AVL tree property is violated.
- Similar to an insertion operation, there are four cases to restore the AVL tree property.
- The only difference from insertion is that after we perform a rotation at x, we may have to perform a rotation at some ancestors of x. ◊ It may involve several rotations.

- Therefore, we must continue to trace the path until we reach the root.
- The time complexity to delete a node is dependent on the height of the tree, which is also O(log(n)).

An Extended Example

Insert 3,2,1,4,5,6,7, 16,15,14

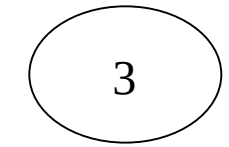


Fig 1

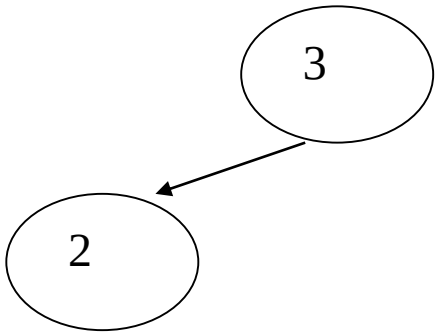


Fig 2

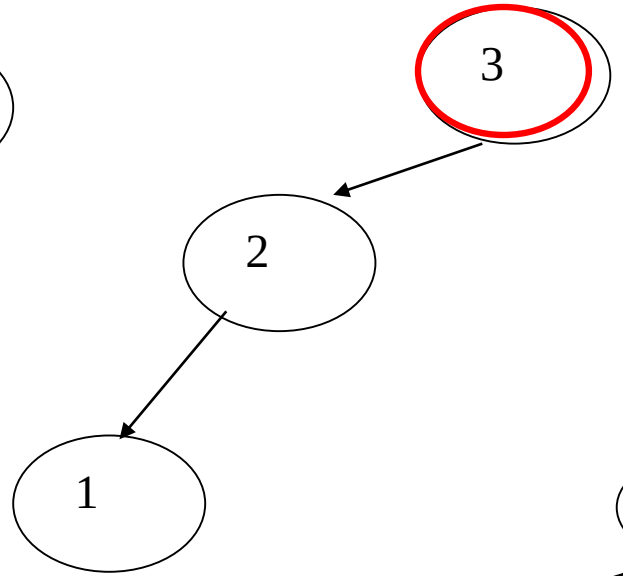


Fig 3

Single rotation

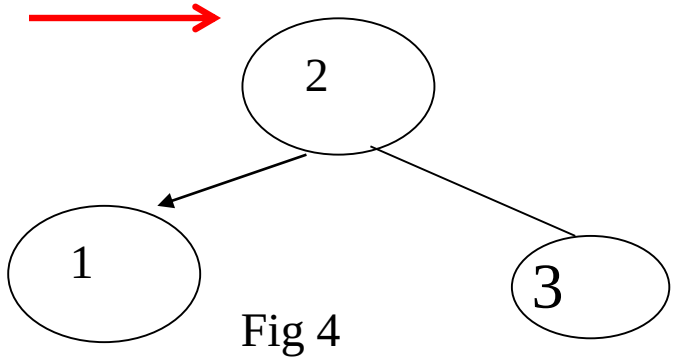


Fig 4

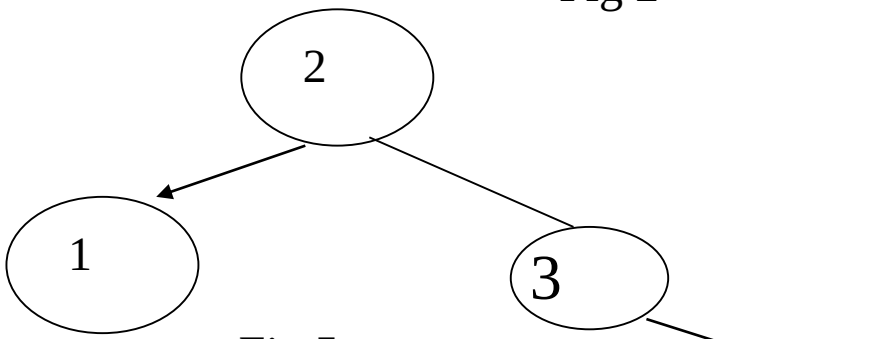
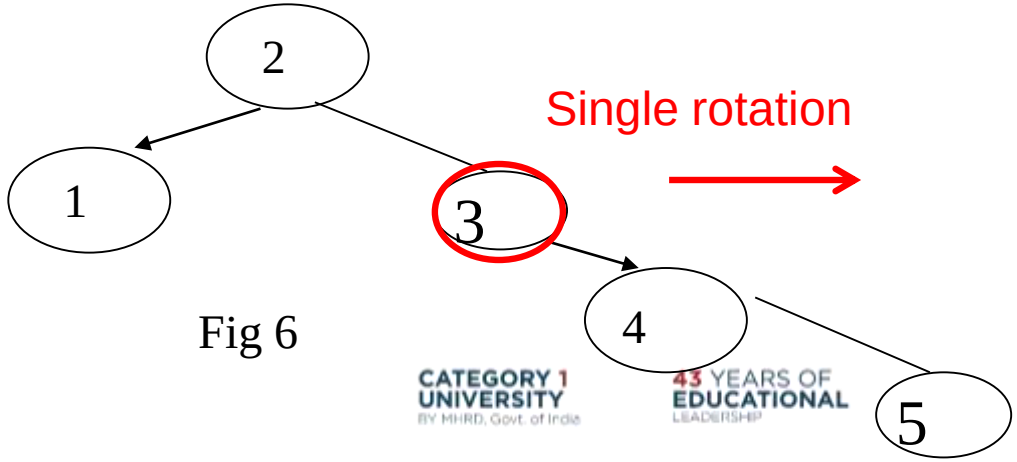


Fig 5



Single rotation

Fig 6

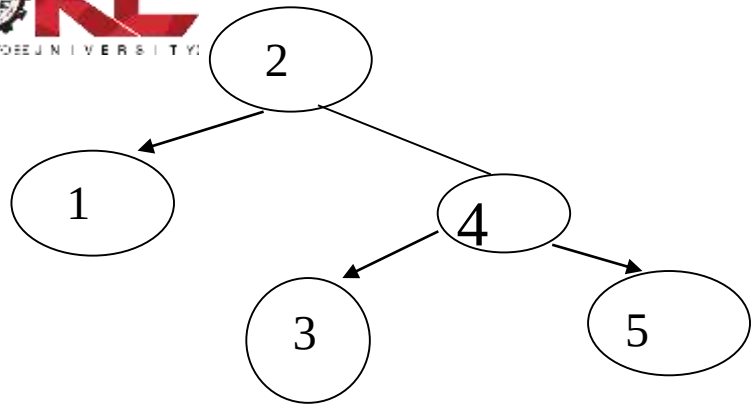
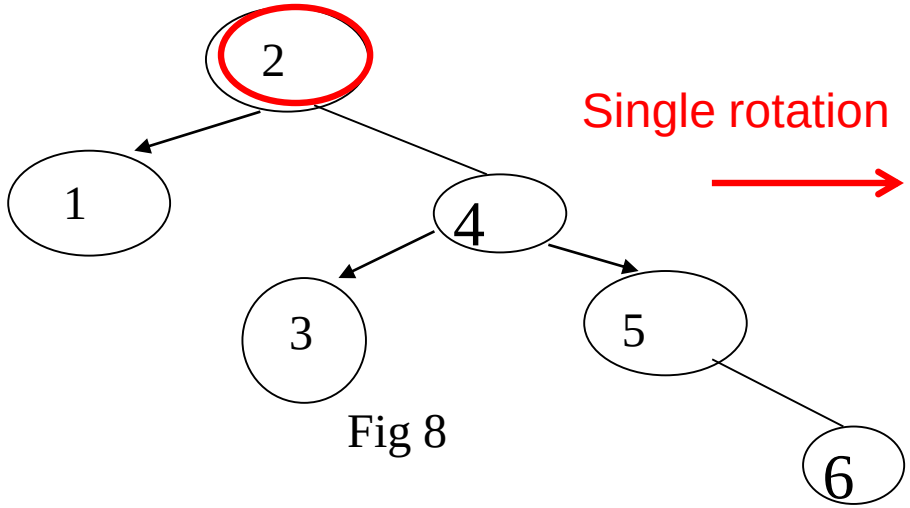


Fig 7



Single rotation

Fig 8

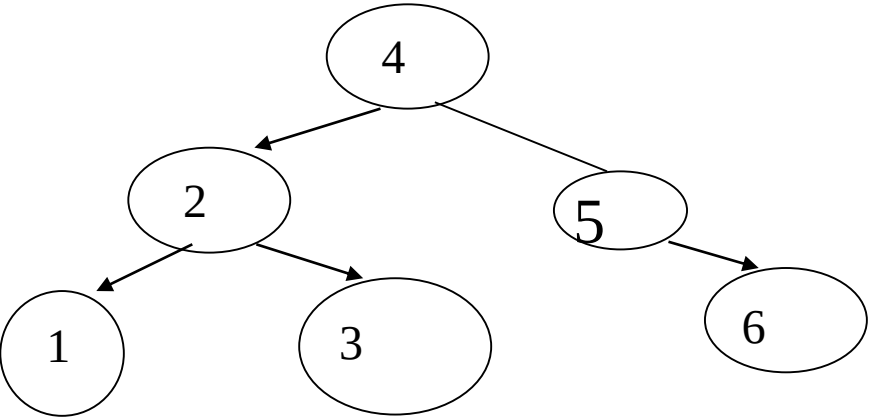
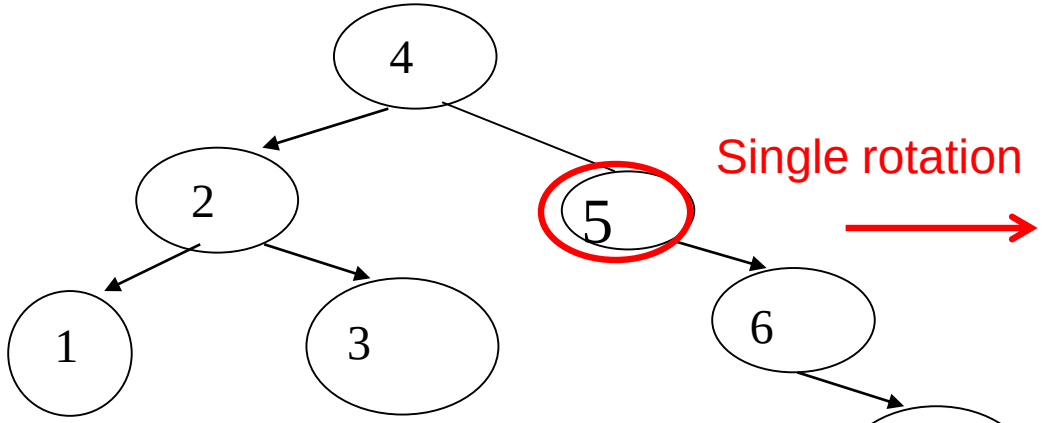


Fig 9



Single rotation

Fig 10

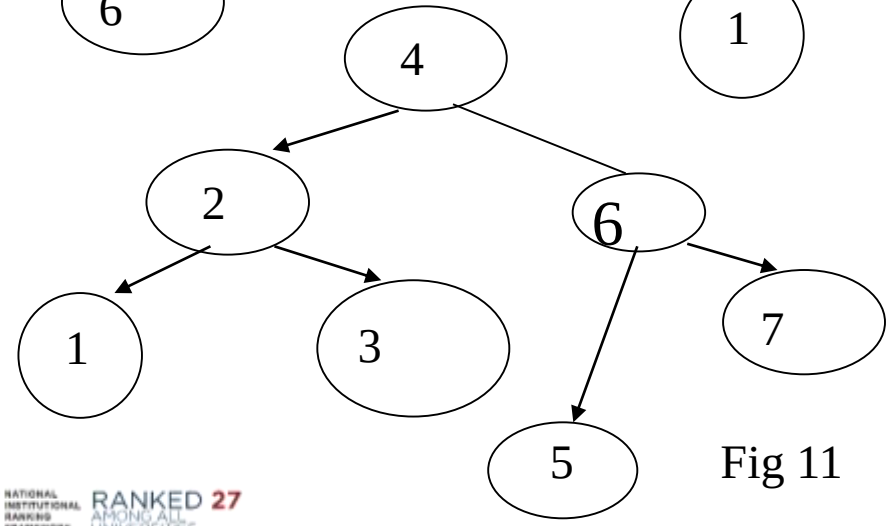


Fig 11

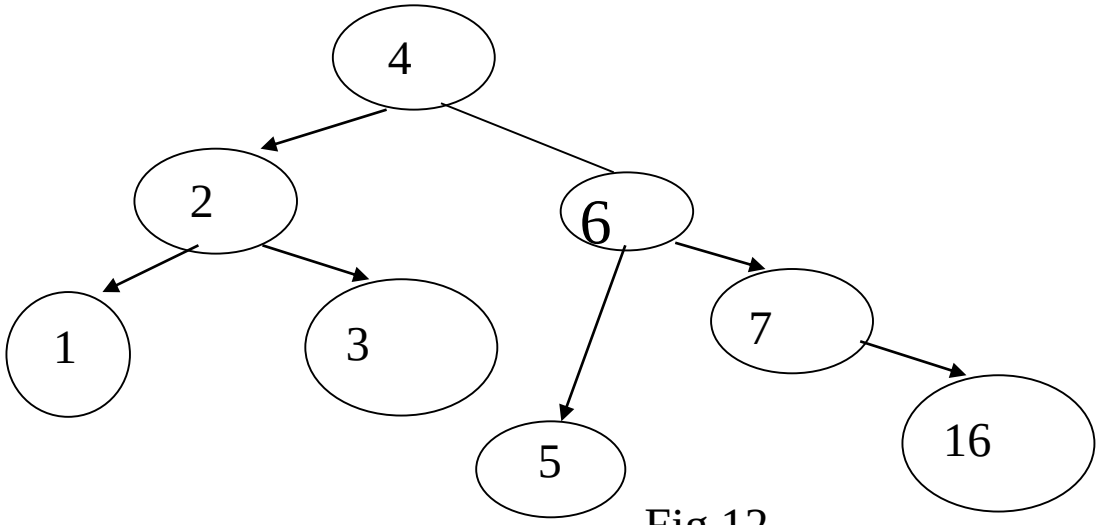


Fig 12

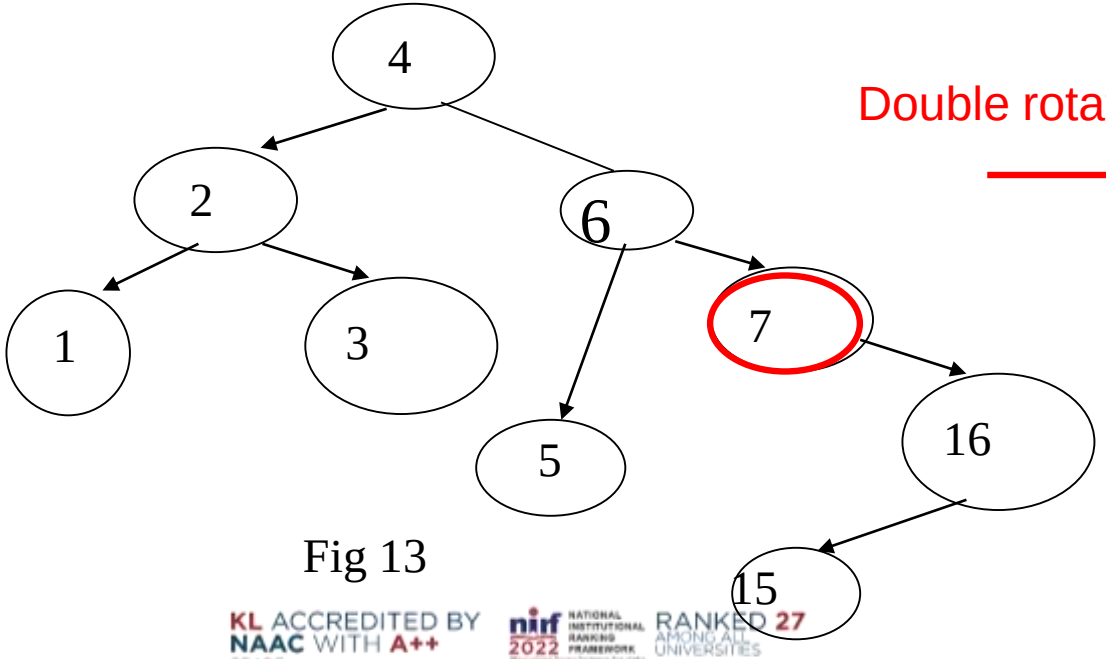


Fig 13

Double rotation

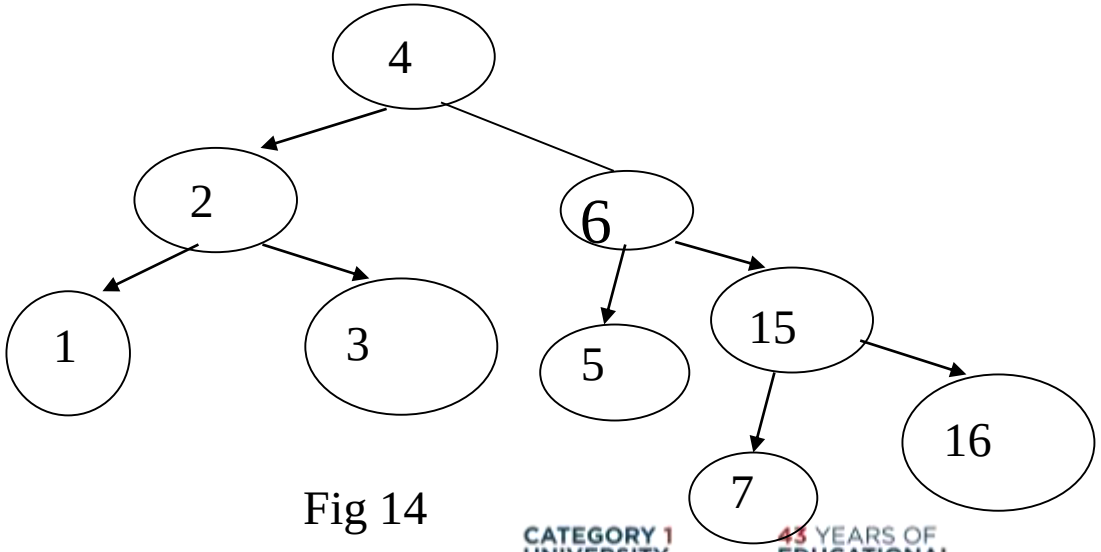


Fig 14

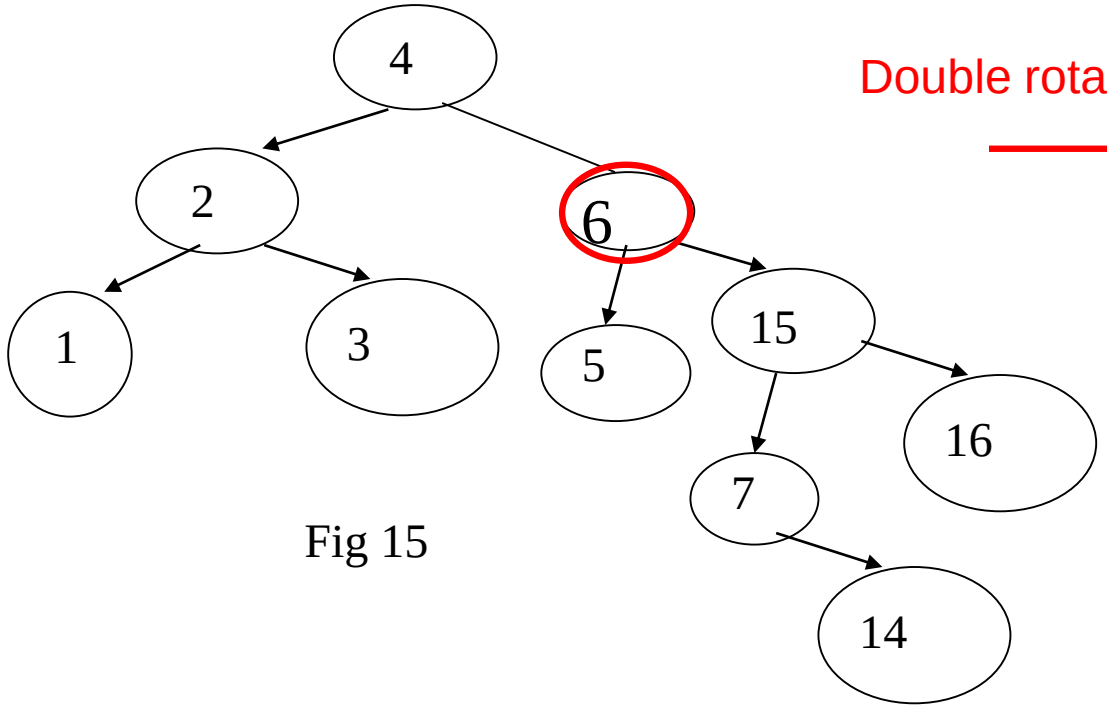


Fig 15

Double rotation

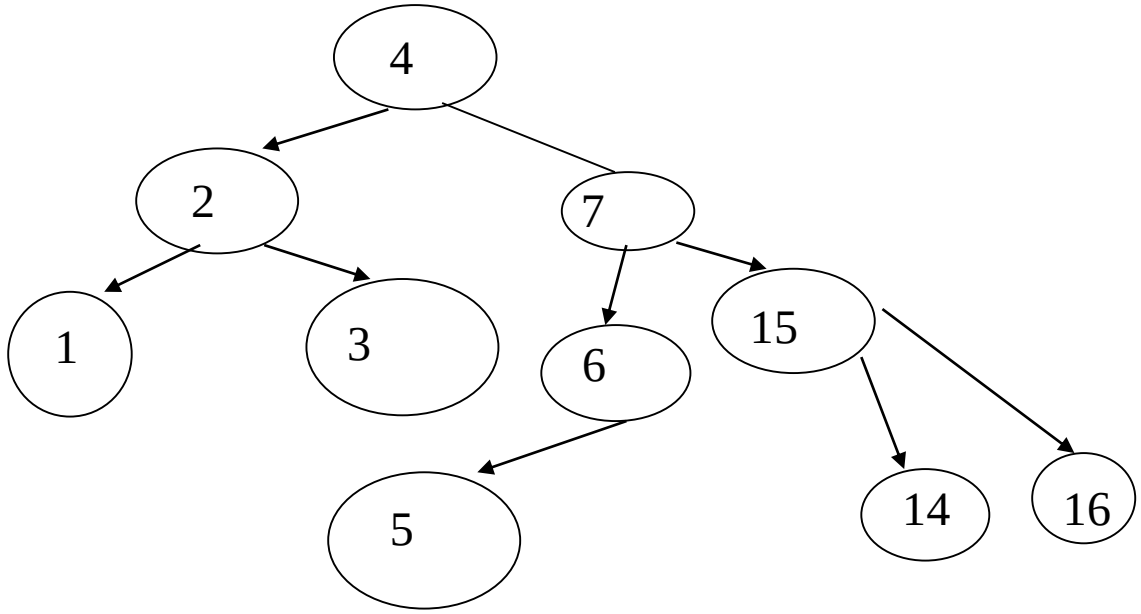


Fig 16

SUMMARY

- AVL Tree is a height-balanced BST that maintains its height by performing tree rotations according to a Balancing Criteria.
- Balancing Criteria: For all nodes, the sub tree height difference (balance factor) should be at most 1
- The valid values of the balance factor are -1, 0, and +1.
- The insert and delete operation require rotations to be performed after violating the balance factor.
- The balance factor of a node is defined as the difference between the height of the left and right sub tree of that node.
- Tree Rotations are changes in the structure of the tree, done only on 3 nodes without affecting the elements' order.

SELF-ASSESSMENT QUESTIONS

1.How to calculate the balance factor of a node in an AVL Tree?

a)height of the left sub tree - height of the right sub tree

b)height of the right sub tree - height of the left sub tree

c) absolute difference between the heights of the left and right sub trees.

d)sum of the heights of the left and right sub trees.

2.What is the primary purpose of rotations in AVL Trees?

a)To increase the overall height of the tree.

b)To maintain a balanced structure and ensure efficient operations.

c)To create a perfectly symmetrical tree shape.

d)To sort the nodes in ascending or descending order.

TERMINAL QUESTIONS

1. Describe About AVL Trees?
2. Explain the concept of balance factor in an AVL tree. What are the acceptable values for a node's balance factor?
3. Describe the four different types of rotations in an AVL tree (left, right, double left, double right). When is each type of rotation used?
4. Given a sequence of elements, construct an AVL tree by inserting them one by one. Explain your choices for insertion order and justify any performed rotations.

REFERENCES FOR FURTHER LEARNING OF THE SESSION

Reference Books:

1. Mark Allen Weiss, Data Structures and Algorithm Analysis in C, 2010 , Second Edition, PearsonEducation.
2. 2. Ellis Horowitz, Fundamentals of Data Structures in C: Second Edition, 2015
3. A.V.Aho, J. E. Hopcroft, and J. D. Ullman, “Data Structures And Algorithms”, Pearson Education, First Edition Reprint2003.

Sites and Web links:

1. https://nptel.ac.in/courses/106102064
2. https://in.udacity.com/course/intro-to-algorithms--cs215
3. https://www.coursera.org/learn/data-structures?action=enroll

THANK YOU