Unit 4: Advance Tree

Dr Ratna Patil

Unit IV: Advanced Trees

- AVL Trees,
- Indexing and Multiway Trees- Indexing, indexing techniques,
- Types of search tree- Multiway search tree,
- B-Tree, B+Tree,
- Trie Tree,
- Splay Tree,
- Red-Black Tree

Dr Ratna Patil

Balancing Binary Search Trees

Many algorithms exist for keeping binary search trees balanced

- Adelson-Velskii and Landis (AVL) trees (height-balanced trees)
- Splay trees and other self-adjusting trees B-trees and other multiway search trees

Dr Ratna Patil

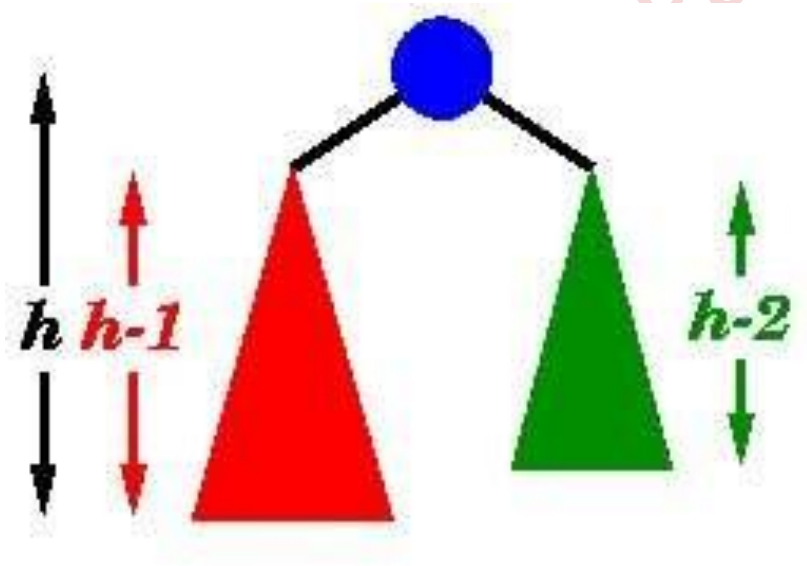
Definition of a balanced tree

- Ensure the depth = O(log N)
- Take O(log N) time for searching, insertion, and deletion
- Every node must have left & right sub-trees of the same height

Dr Ratna Patil

An AVL tree has the following properties:

1. Sub-trees of each node can differ by at most 1 in their height
2. Every sub-trees is an AVL tree

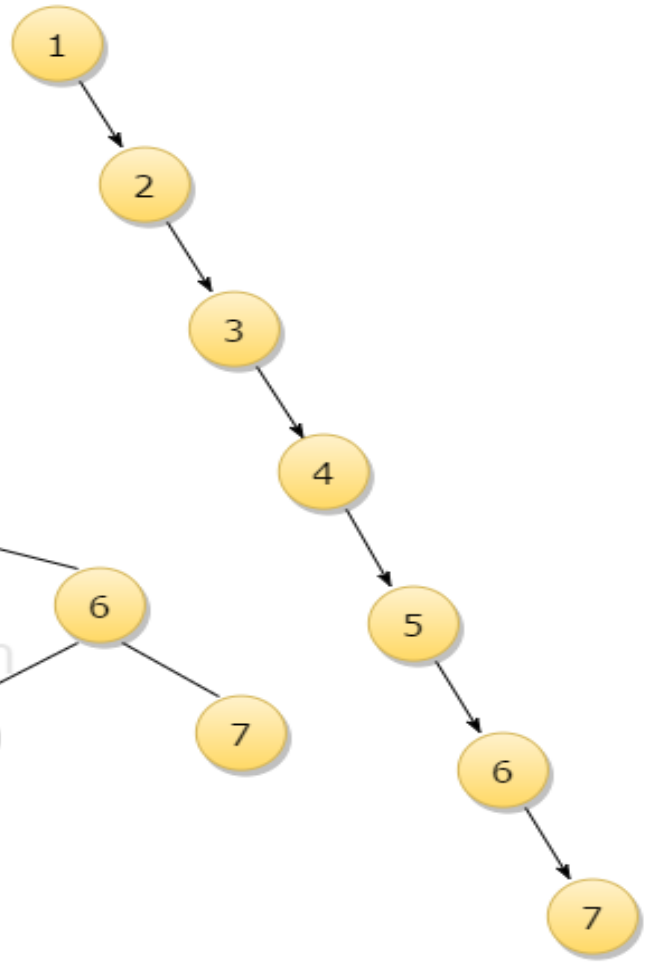


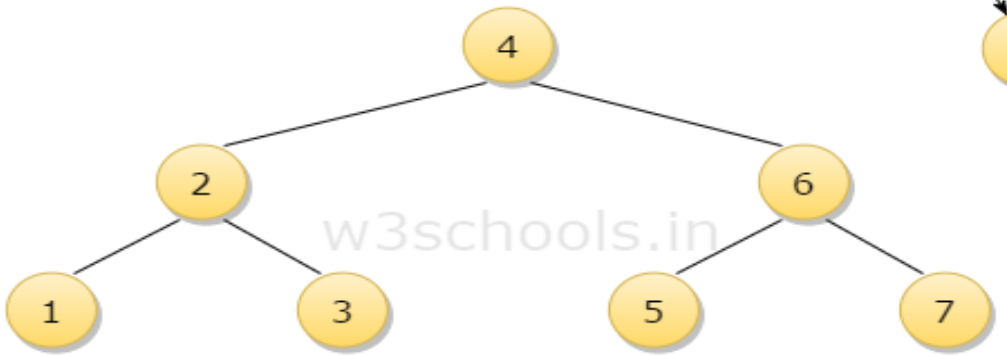
patil

Why AVL Tree?

Example: Keys are: 1, 2, 3, 4, 5, 6, 7. Generate Binary Tree.

- **Fig 1:** AVL Tree
- **Fig 2**. Binary Tree





- **Insert 8 in BT**: <u>7 comparisons</u>.
- **Insert 8 in AVL**: <u>3 comparisons.</u>

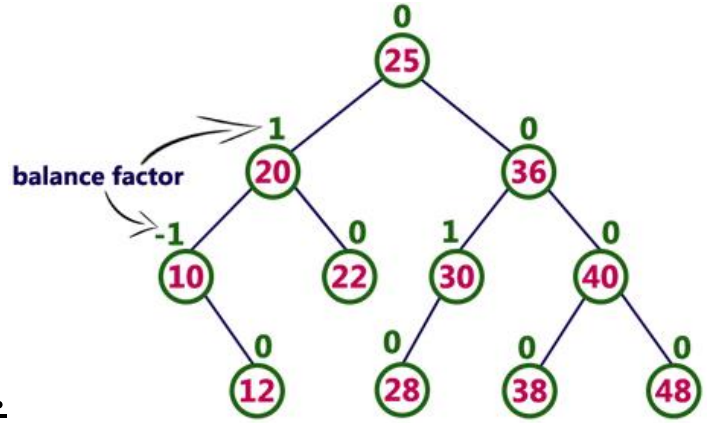
Dr Ra
w3schools.in

So, AVL Tree:

- **Height balance trees.**
- **Insertion and deletion have low time complexity**.

What is AVL Tree?

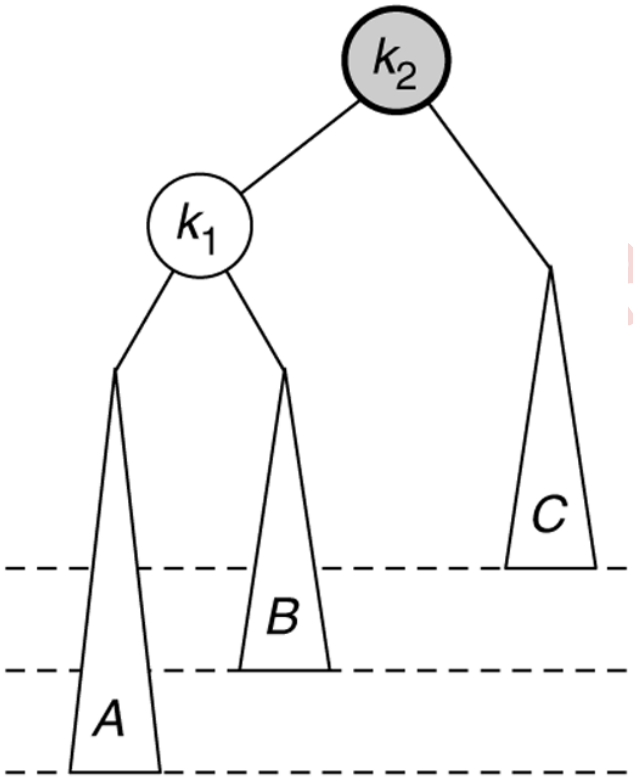
- AVL invented by G.M. Adelson-Velsky and E.M. Landis. So, name is AVL.
- AVL tree is a height-balanced binary search tree.
- AVL tree, balance factor of every node is either -1, 0 or +1.
- Every node maintains an extra information known as balance factor.
- **Balance factor = heightOfLeftSubtree – heightOfRightSubtree.**
- Every AVL Tree is a binary search tree but every Binary Search Tree need not be AVL tree.
- Operation perform Search, Insertion, Deletion with **O(log n)** time complexity.

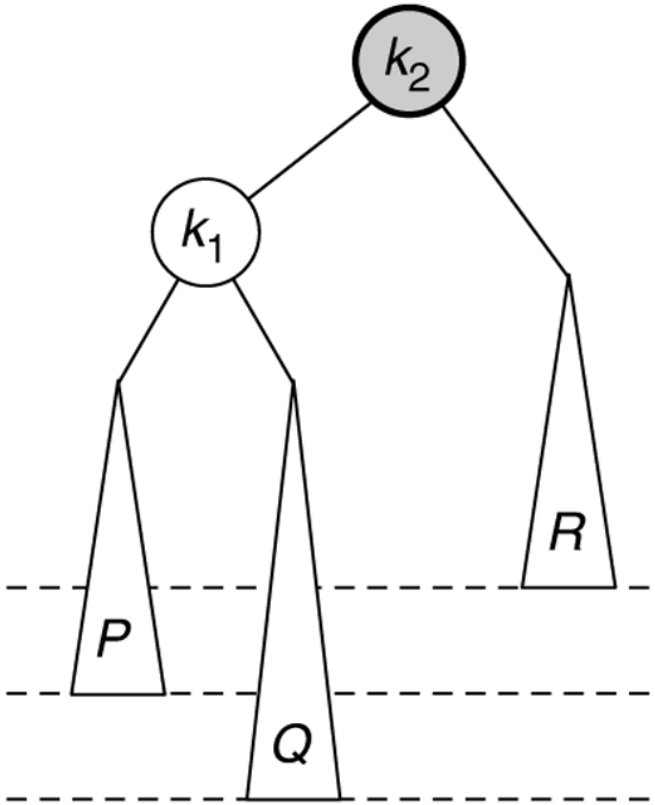


Dr Ratna Patil

Problem Cases for AVL insert

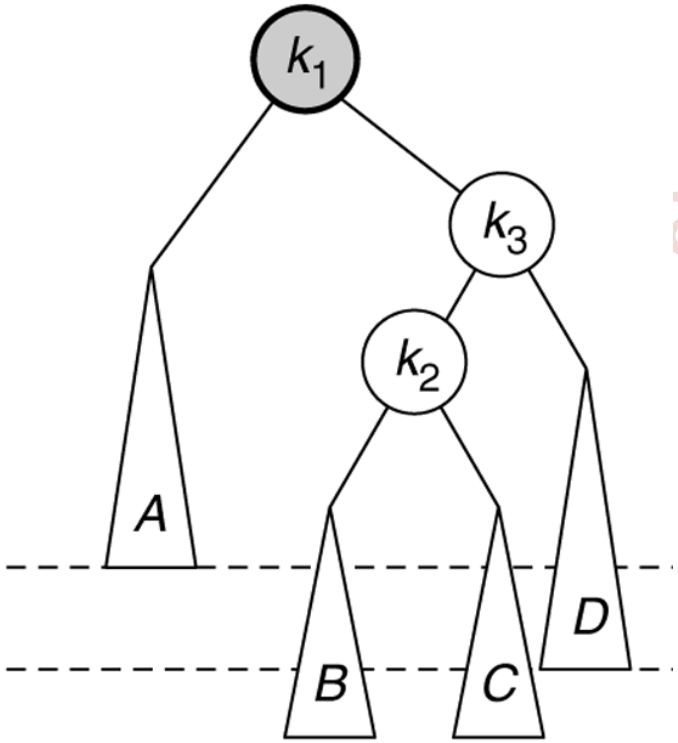
1. LL Case: insertion into left subtree of node's left child
2. LR Case: insertion into right subtree of node's left child

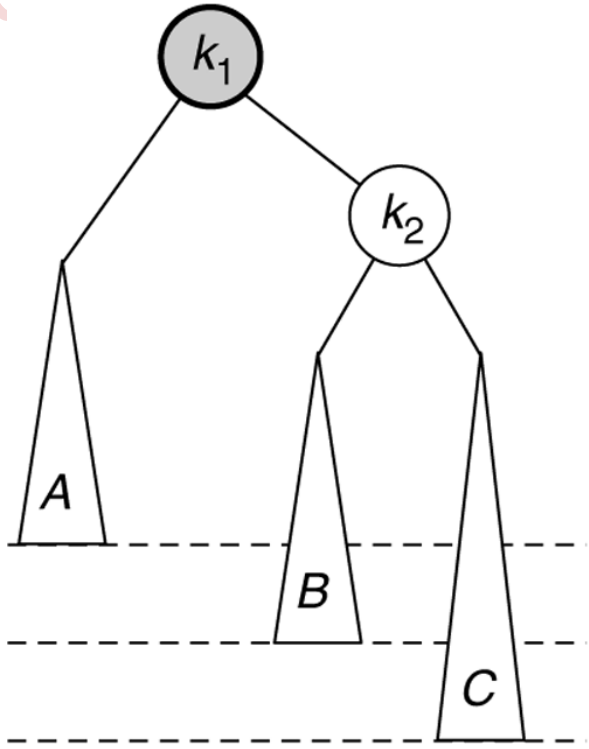




Problem Cases for AVL insert (cont'd)

3. RL Case: insertion into left subtree of node's right child
4. RR Case: insertion into right subtree of node's right child

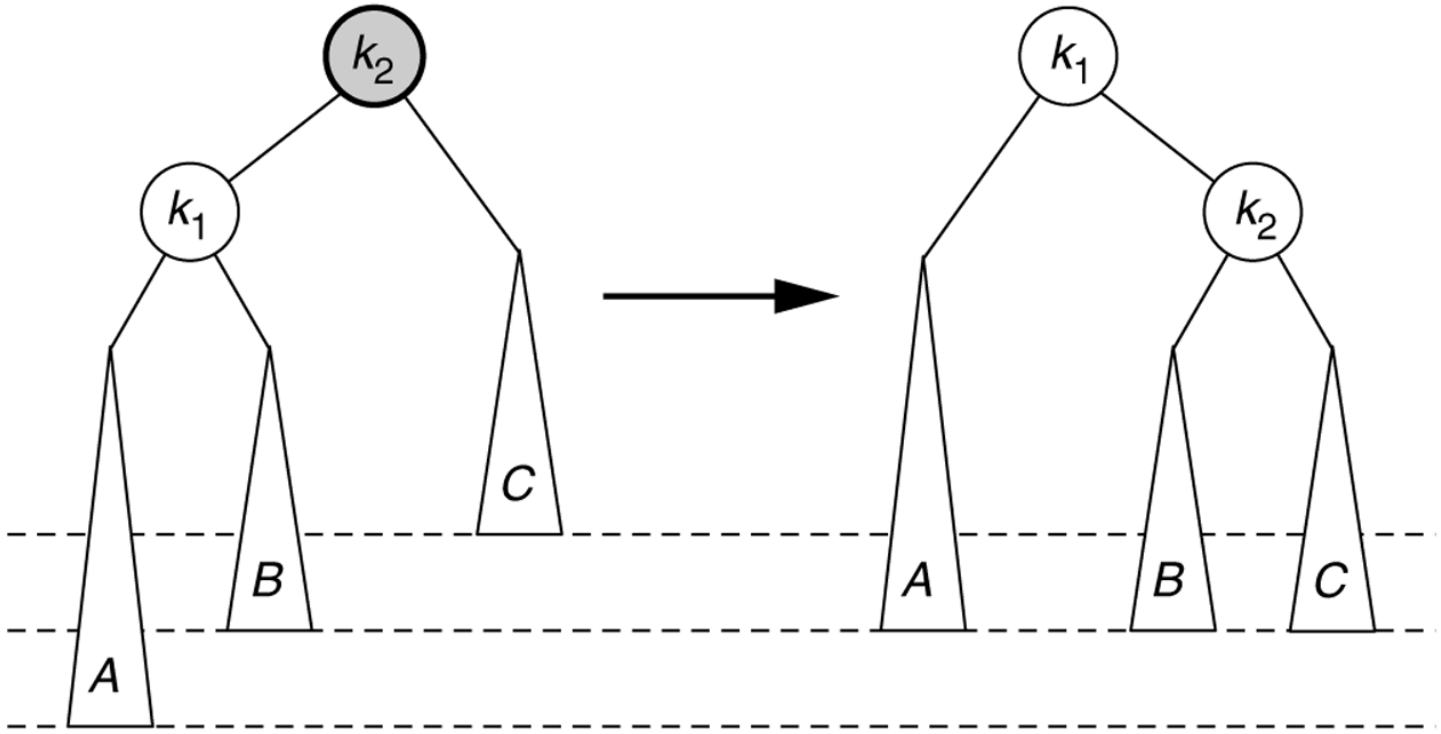




atna Patil

Right rotation to fix Case 1 (LL)

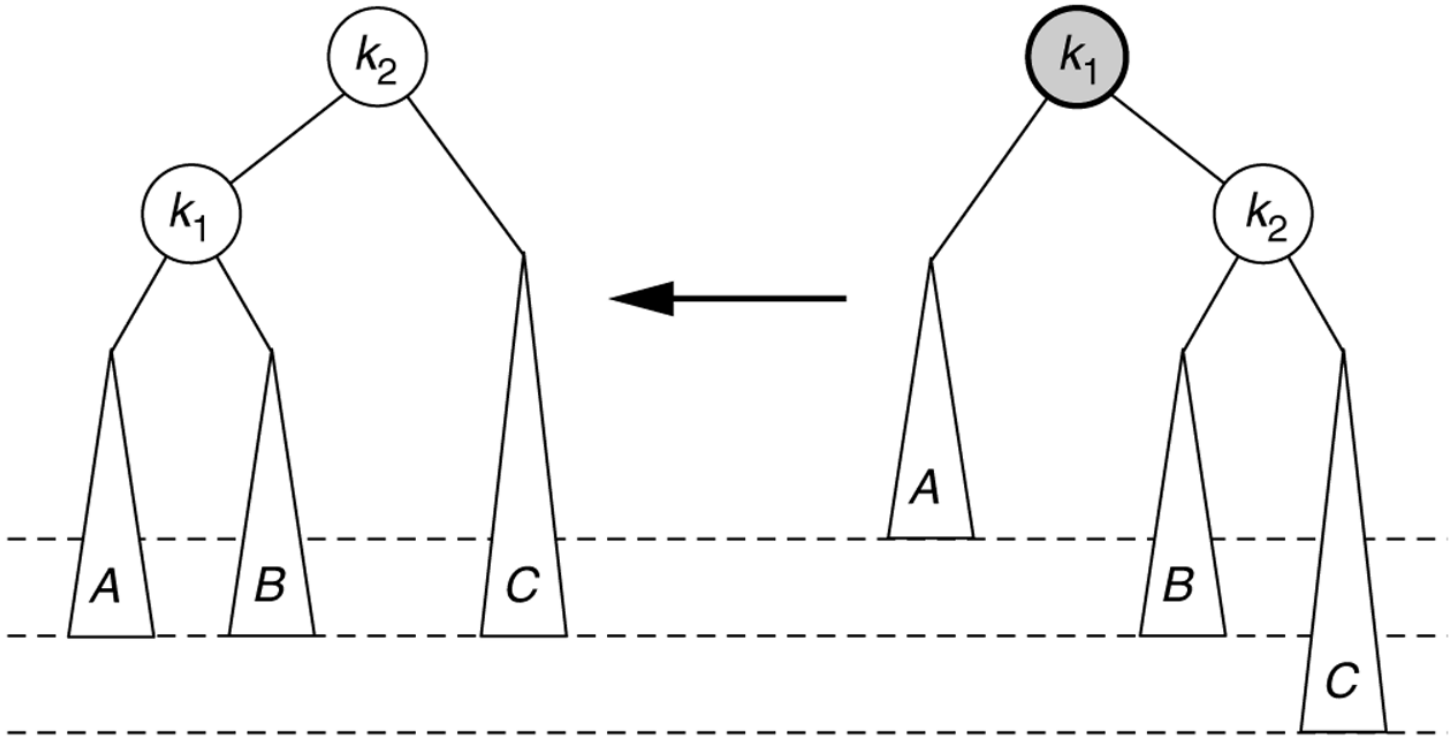
- **right rotation** (clockwise): left child becomes parent; original parent demoted to right



(a) Before rotation (b) After rotation

Left rotation to fix Case 4 (RR)

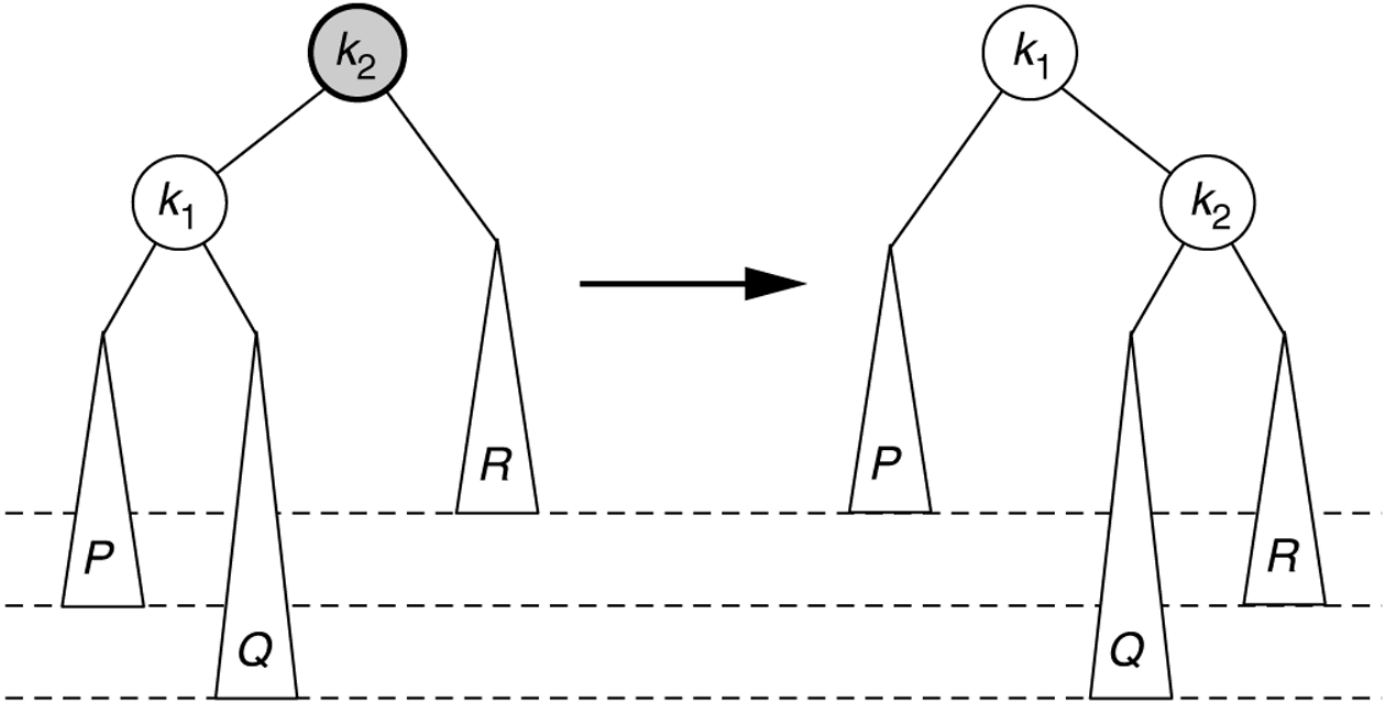
- **left rotation** (counter-clockwise): right child becomes parent; original parent demoted to left



(a) After rotation (b) Before rotation

Problem: Cases 2, 3

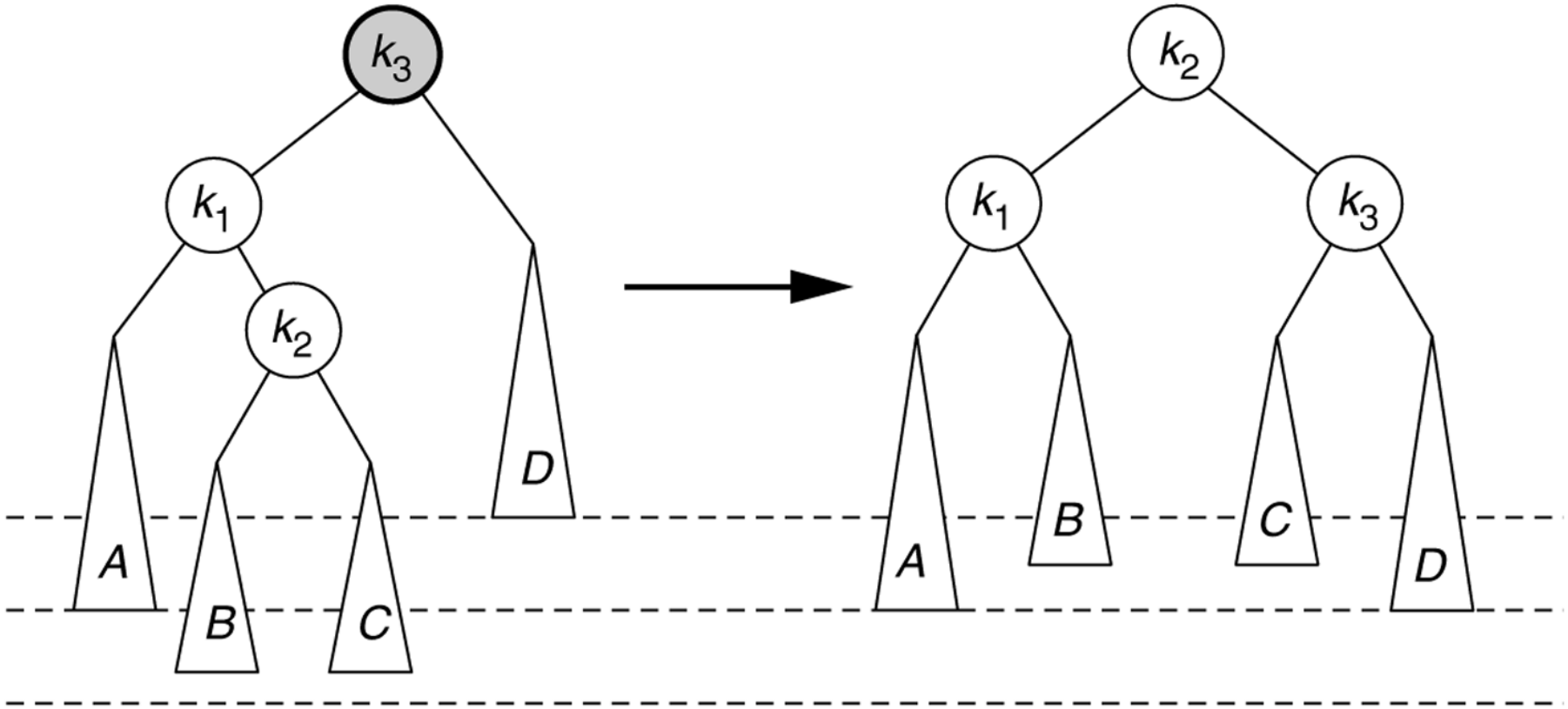
- A single right rotation does not fix Case 2!
- Similarly, a single left rotation does not fix Case 3!



(a) Before rotation (b) After rotation

Left-right rotation for Case 2

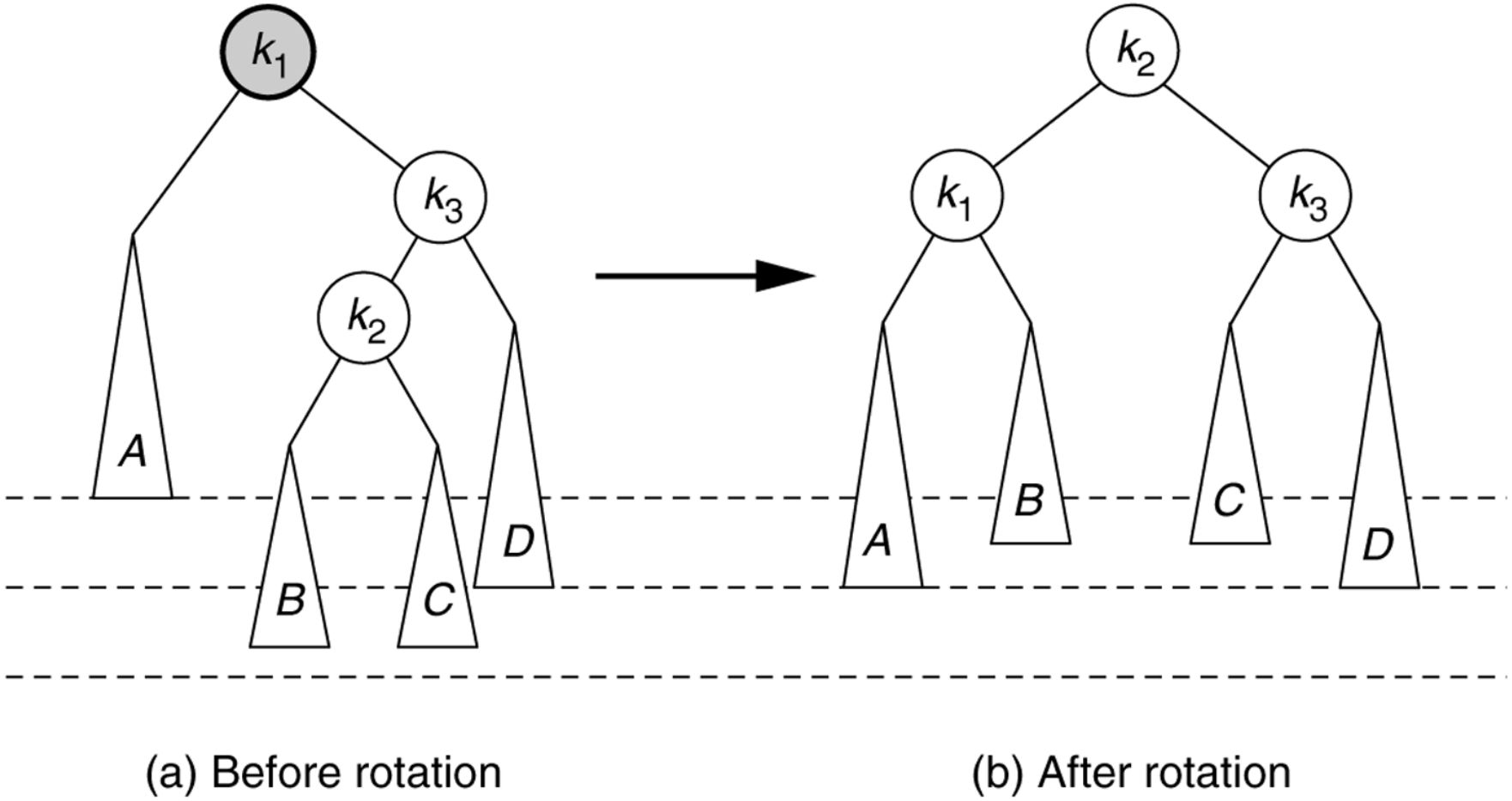
- **left-right double rotation**: a left rotation of the left child, followed by a right rotation at the parent



(a) Before rotation (b) After rotation

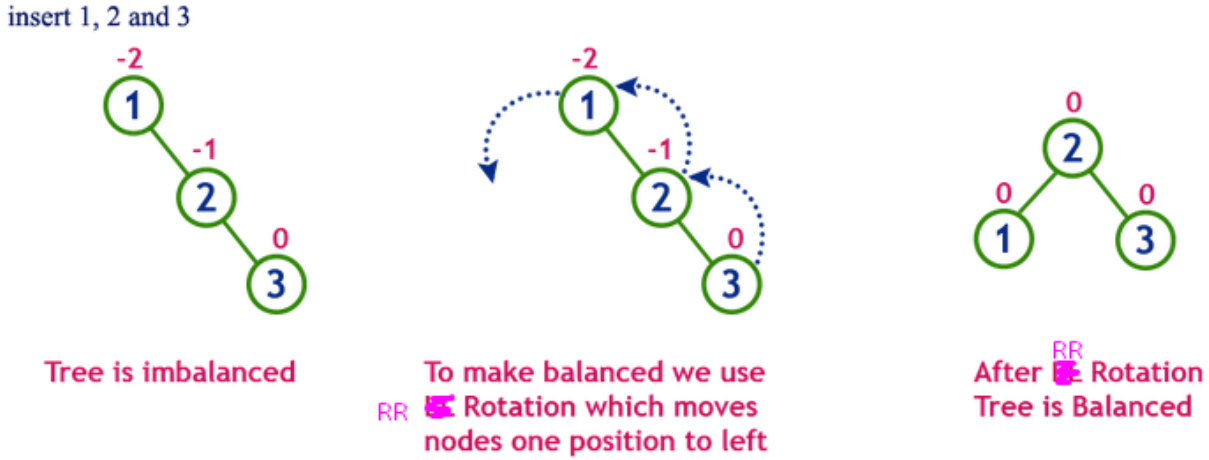
Right-left rotation for Case 3

- **right-left double rotation**: a right rotation of the right child, followed by a left rotation at the parent

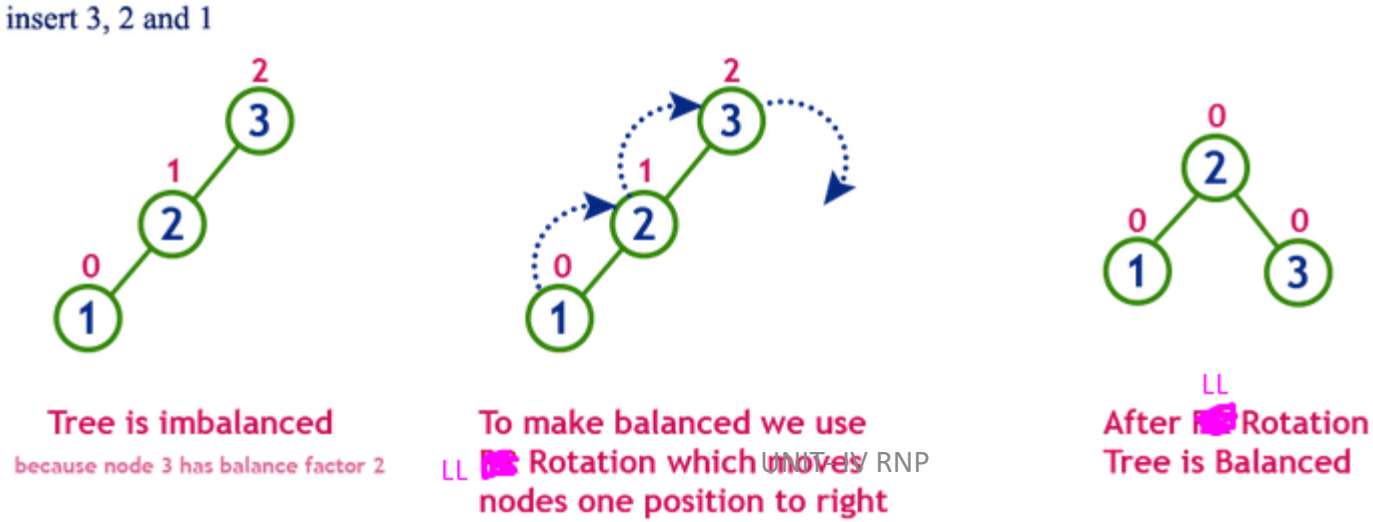


AVL Tree Rotations:

<u>Type 1</u>: RR Relationship (Left Rotation):

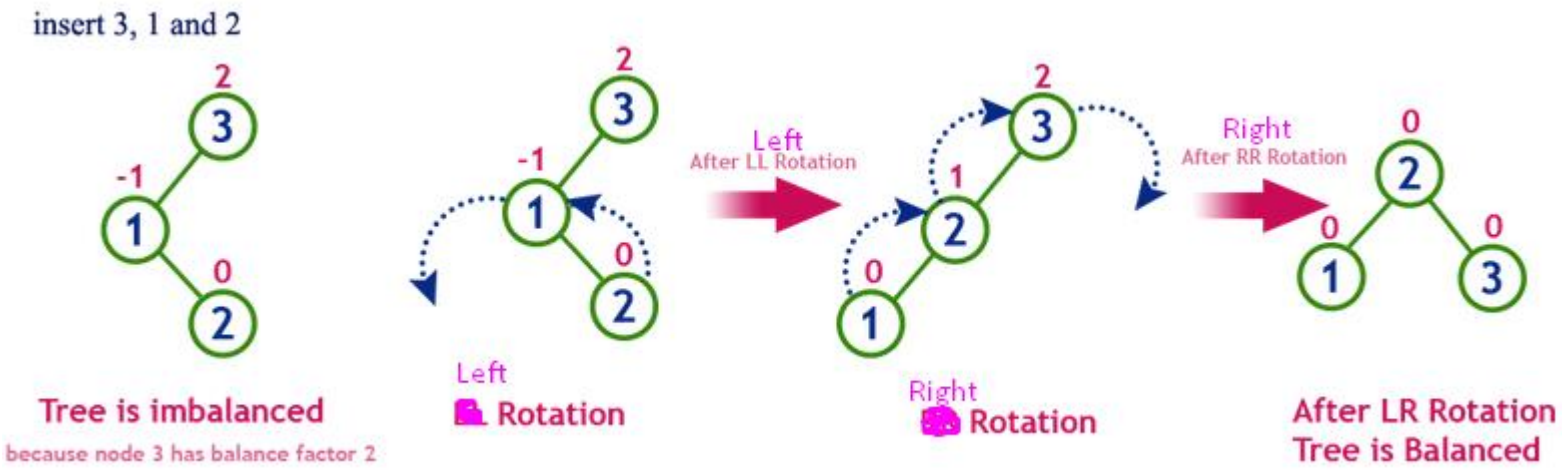


<u>Type 2:</u> LL Relationship (Right Rotation)

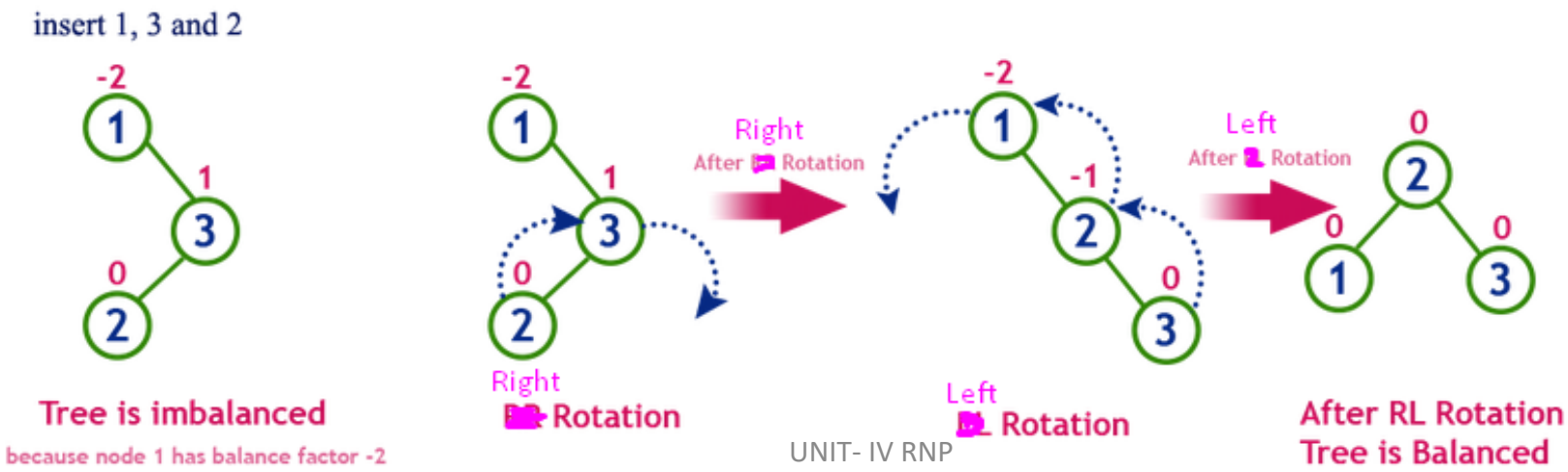


AVL Tree Rotations:

<u>Type 3</u>: LR Relationship (First Left Rotation then Right Rotation)



<u>Type 4:</u> RL Relationship (First Right Rotation then Left Rotation)



Insertion Operations In AVL Tree

In AVL Tree, a new node is always inserted as a leaf node.

Step 1 - Insert the new element into the tree using Binary Search Tree insertion logic.

Step 2 - After insertion, check the **Balance Factor** of every node.

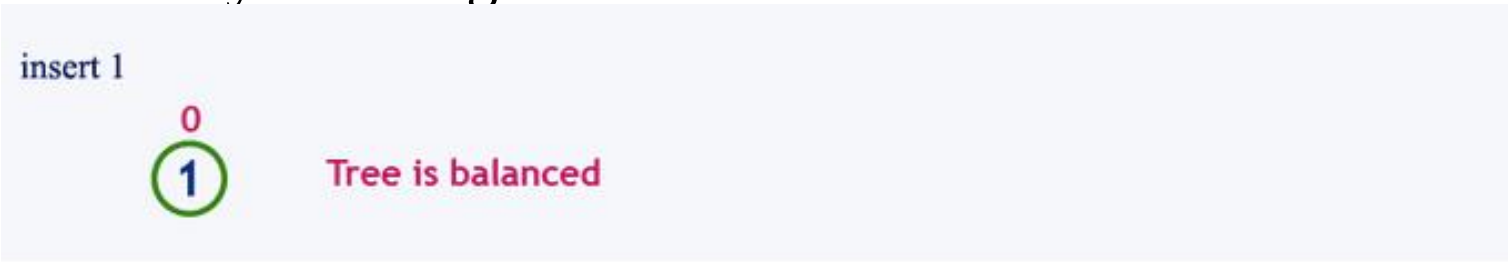
Step 3 - If the **Balance Factor** of every node is **0 or 1 or -1** then go for next operation.

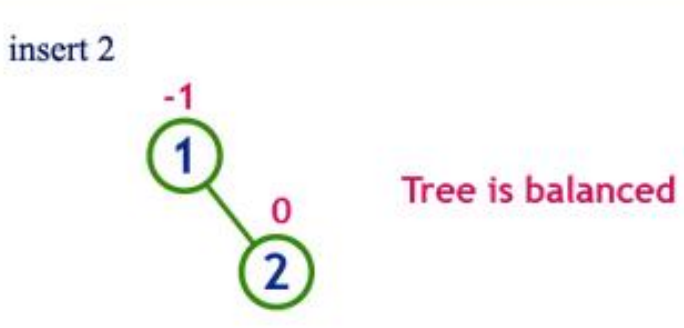
Step 4 - If the **Balance Factor** of any node is other than **0 or 1 or -1** then that tree is said to be imbalanced. In this case, perform suitable **Rotation** to make it balanced and go for next operation.

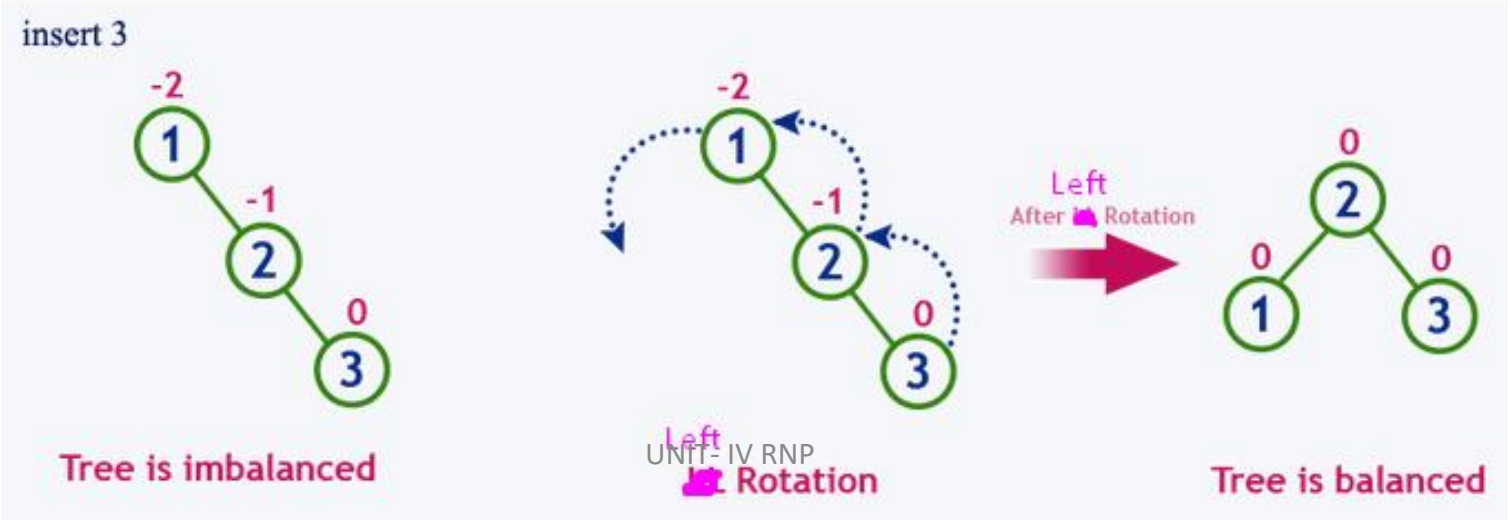
Dr Ratna Patil

Example of Insertion Operation in AVL Tree

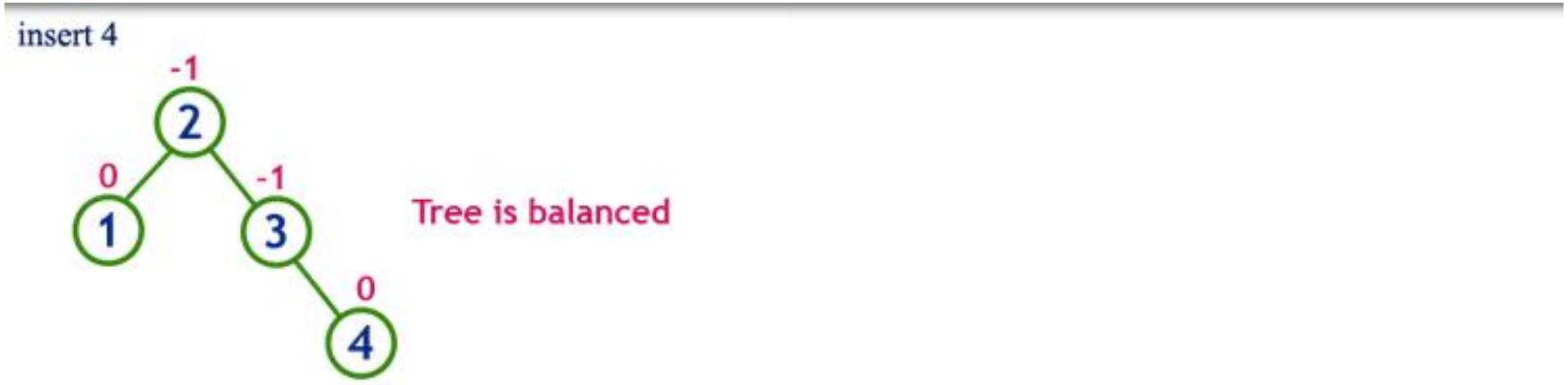
Que. Construct AVL Tree by inserting number from 1 to 8.

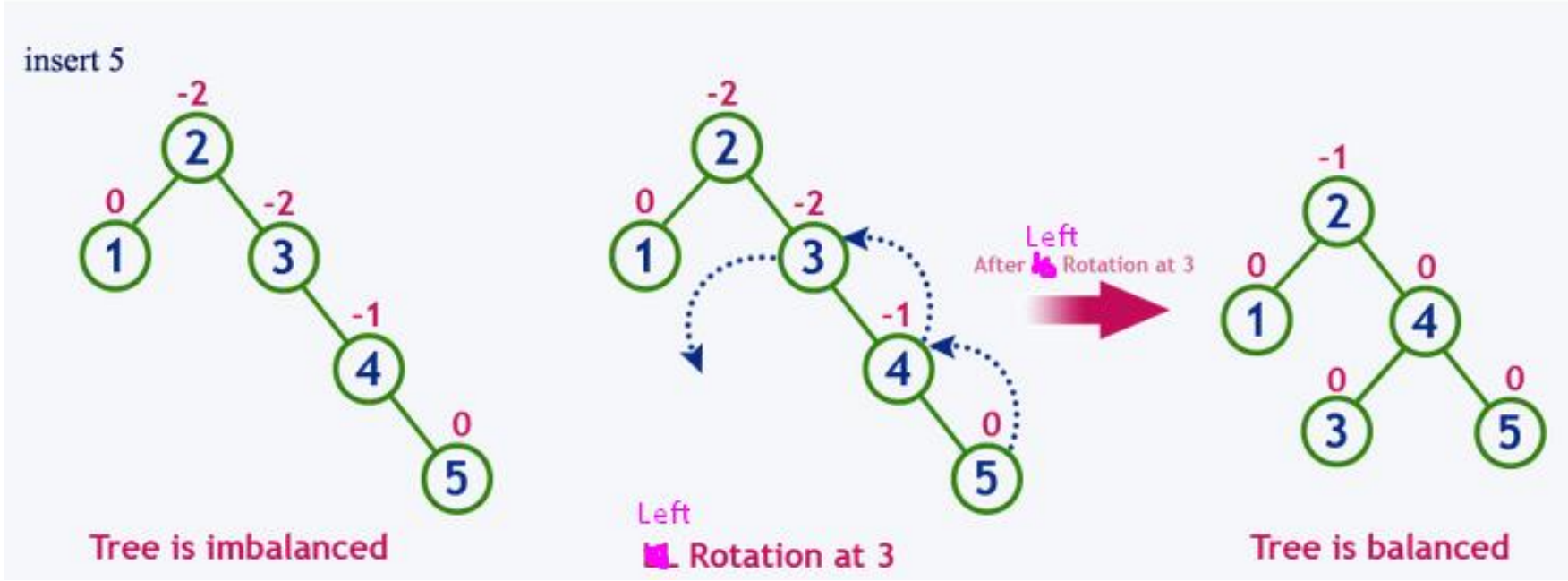




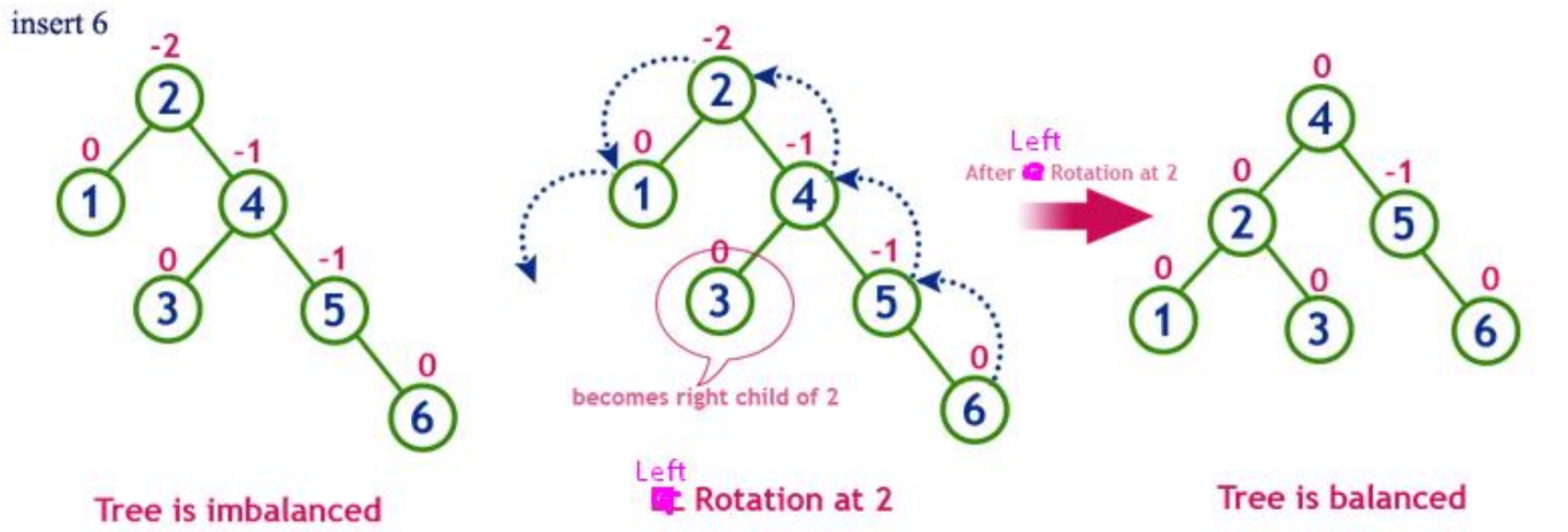


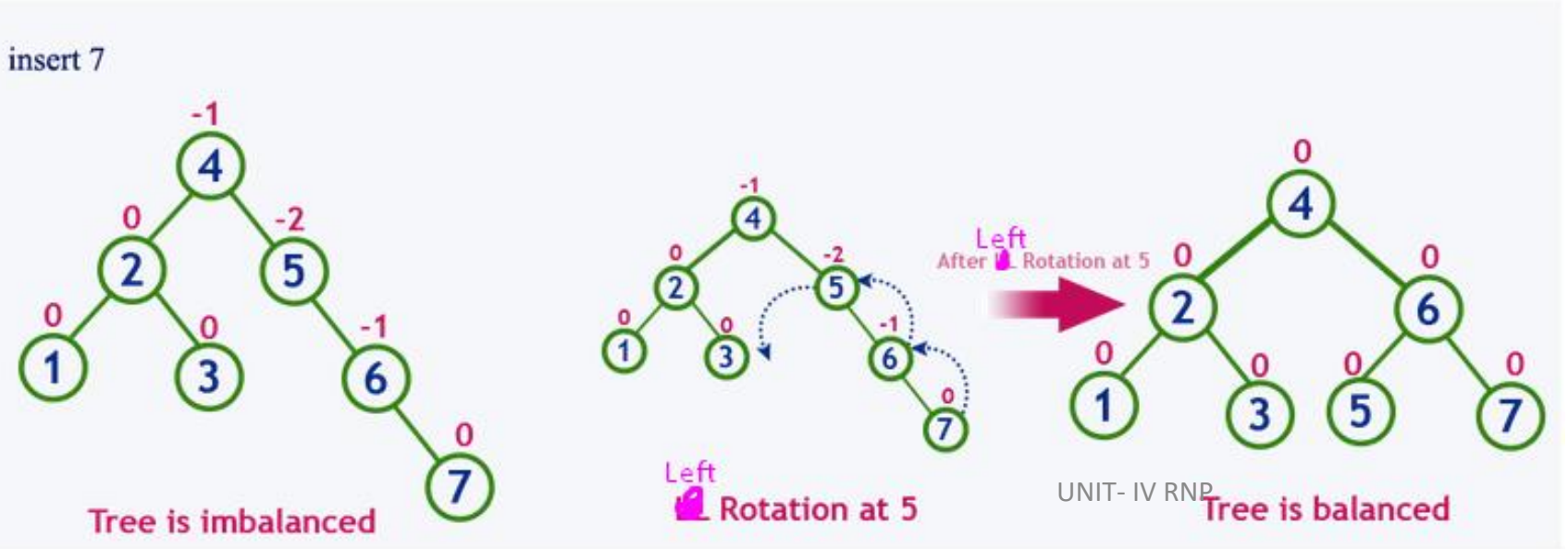
Continue..

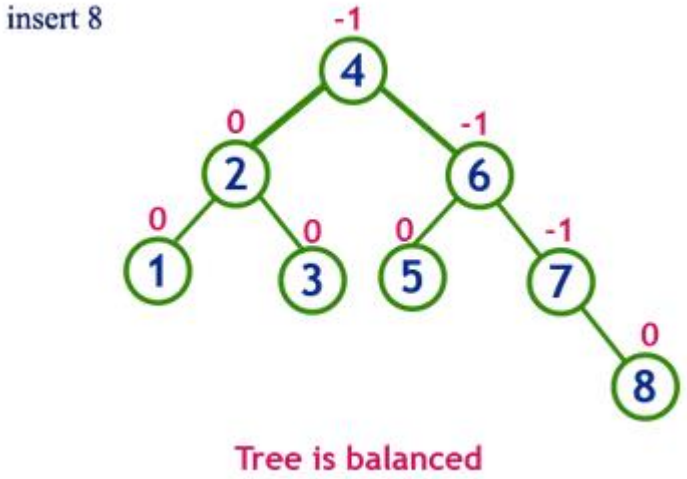

insert 4
-1
2
0
-1
1
3
0
4
Tree is balanced


insert 5
-2
2
0
-2
1
3
-1
4
0
5
Tree is imbalanced
-2
2
0
-2
1
3
-1
4
0
5
Left
Rotation at 3
Left
After Rotation at 3
-1
2
0
0
1
4
0
0
3
5
Tree is balanced

Continue..

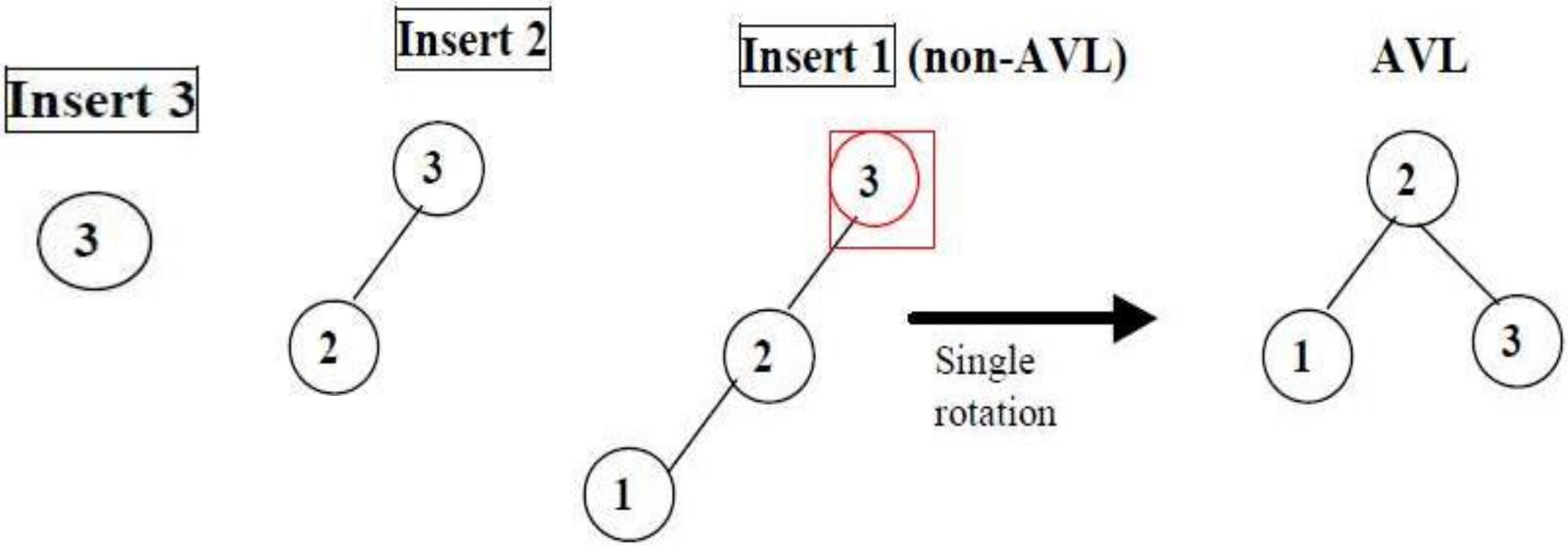

insert 6
Tree is imbalanced
becomes right child of 2
Left
Rotation at 2
Left
After Rotation at 2
Tree is balanced


insert 7
Tree is imbalanced
Left
Rotation at 5
Left
After Rotation at 5
Tree is balanced

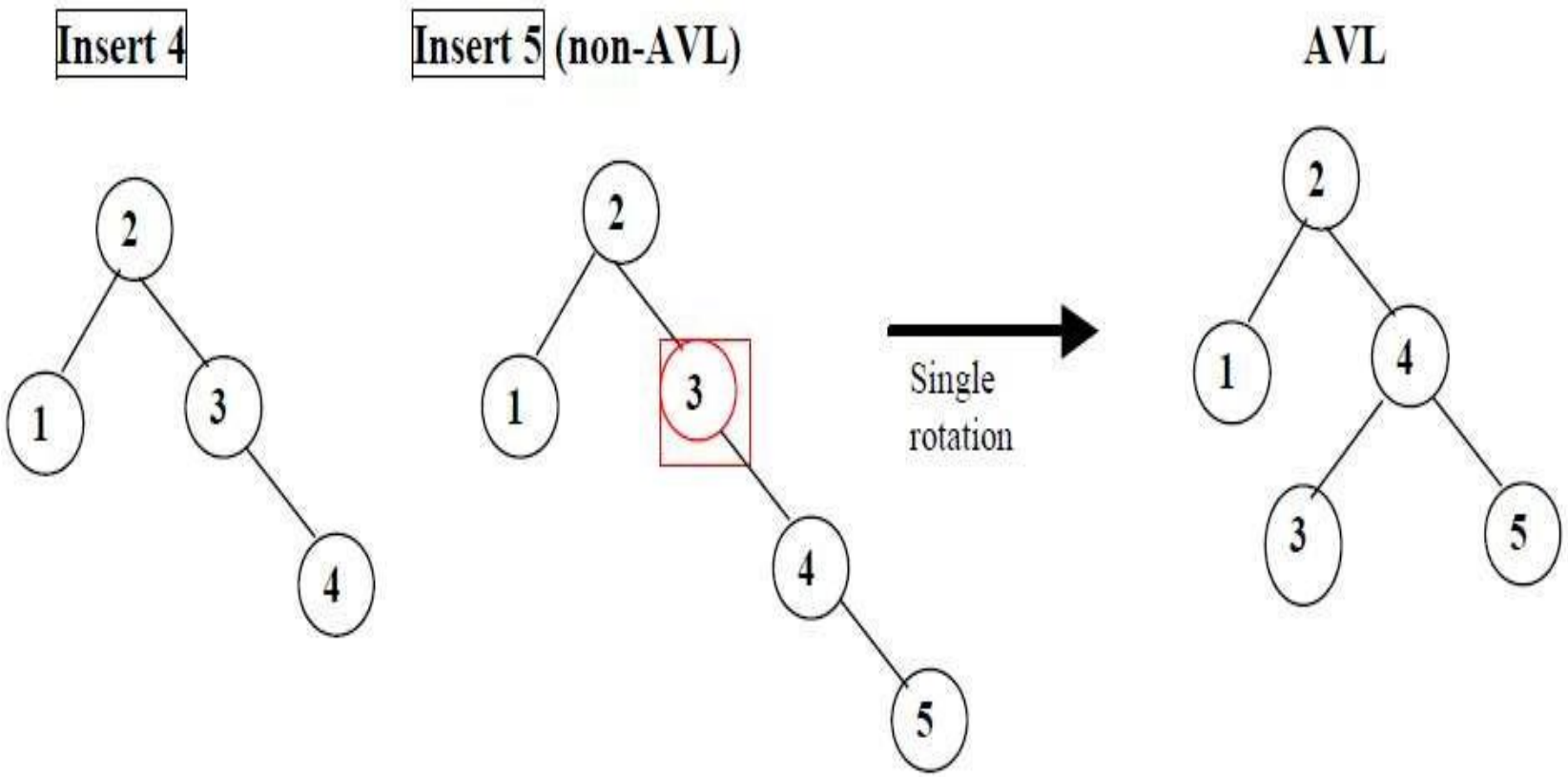

insert 8
Tree is balanced

AVL Trees Example

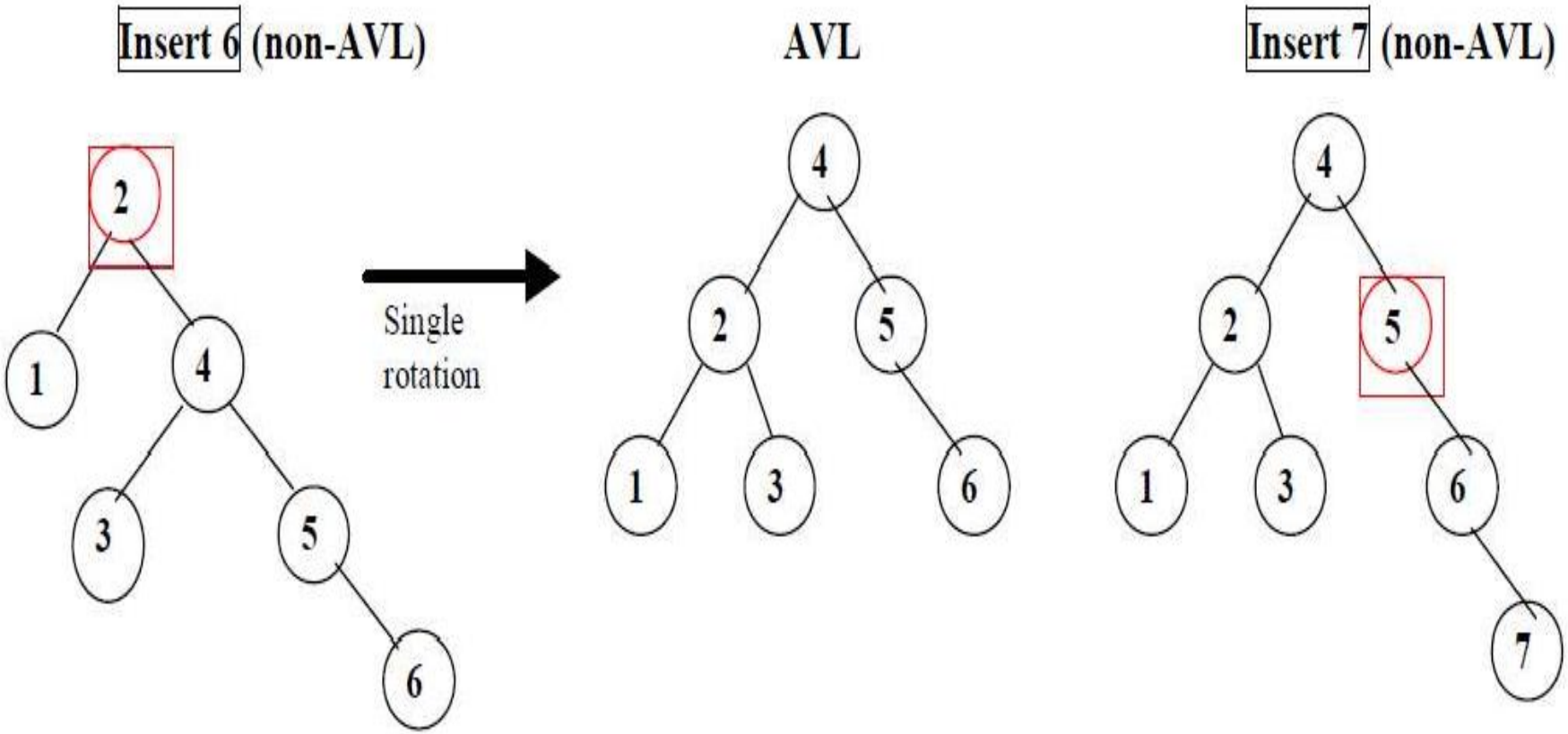
3,2,1,4,5,6,7,16,15,14



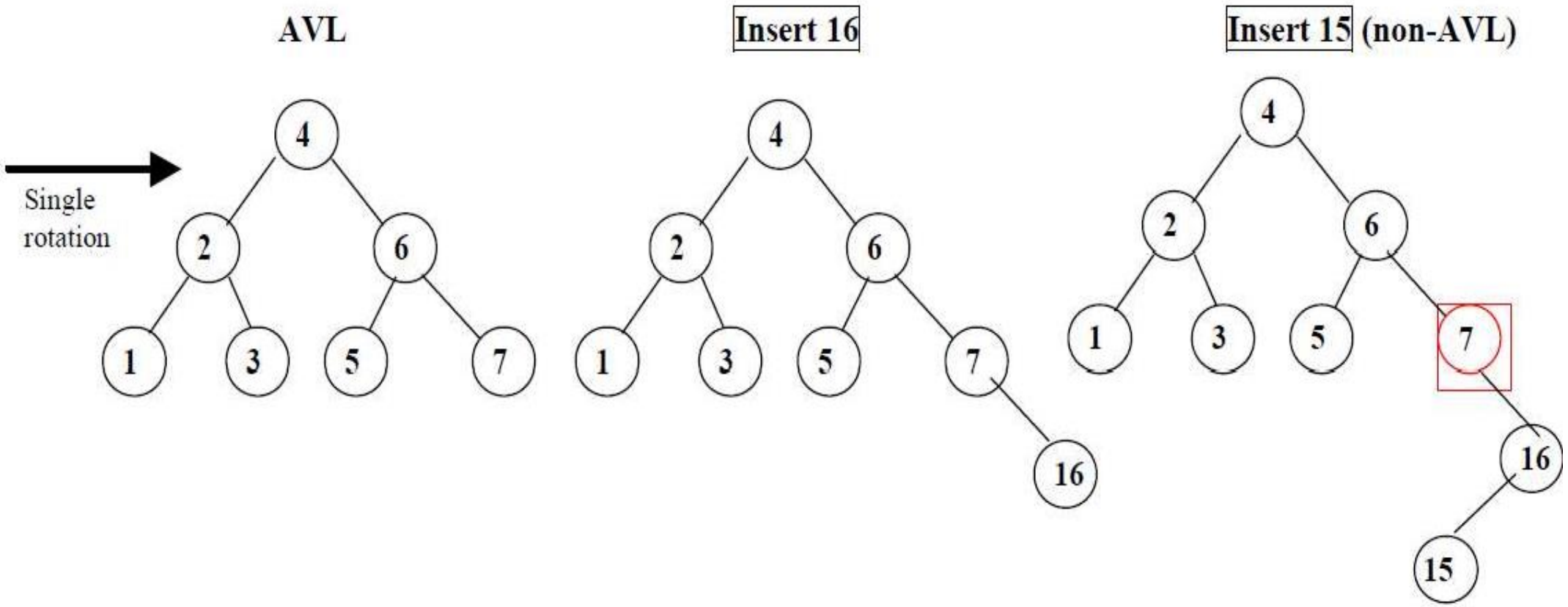
AVL Trees Example



AVL Trees Example



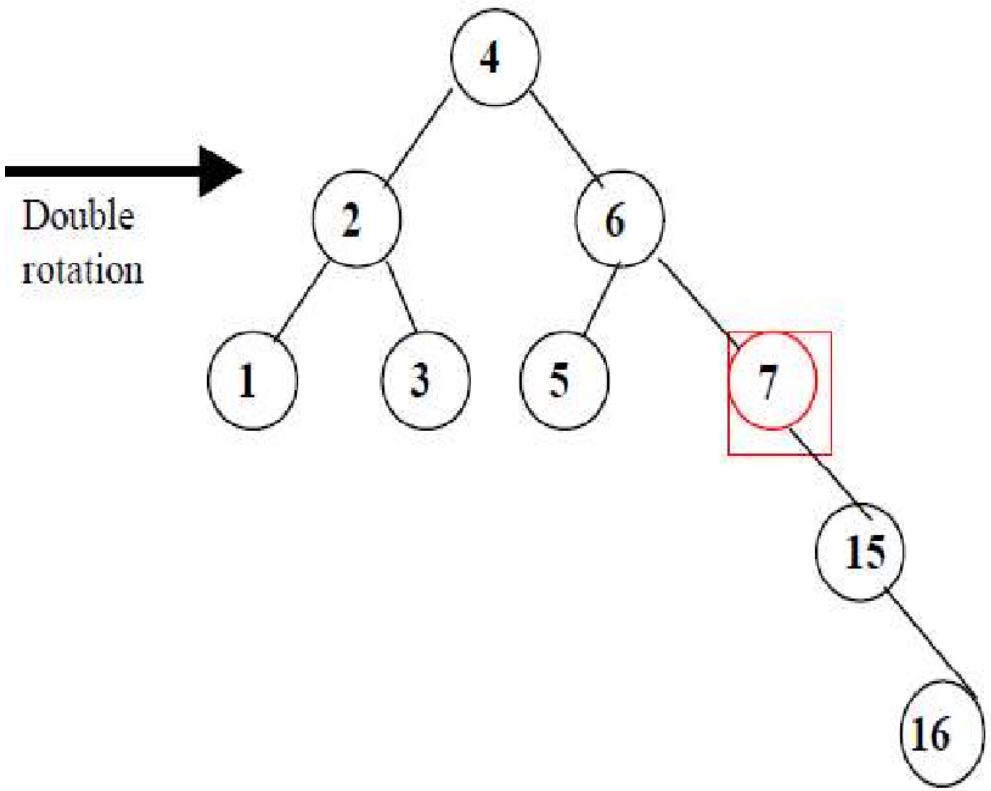
AVL Trees Example

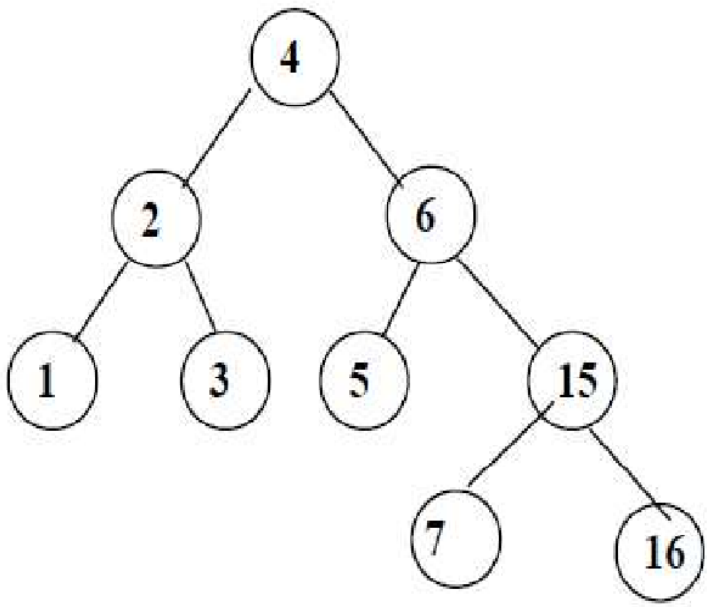


AVL Trees Example

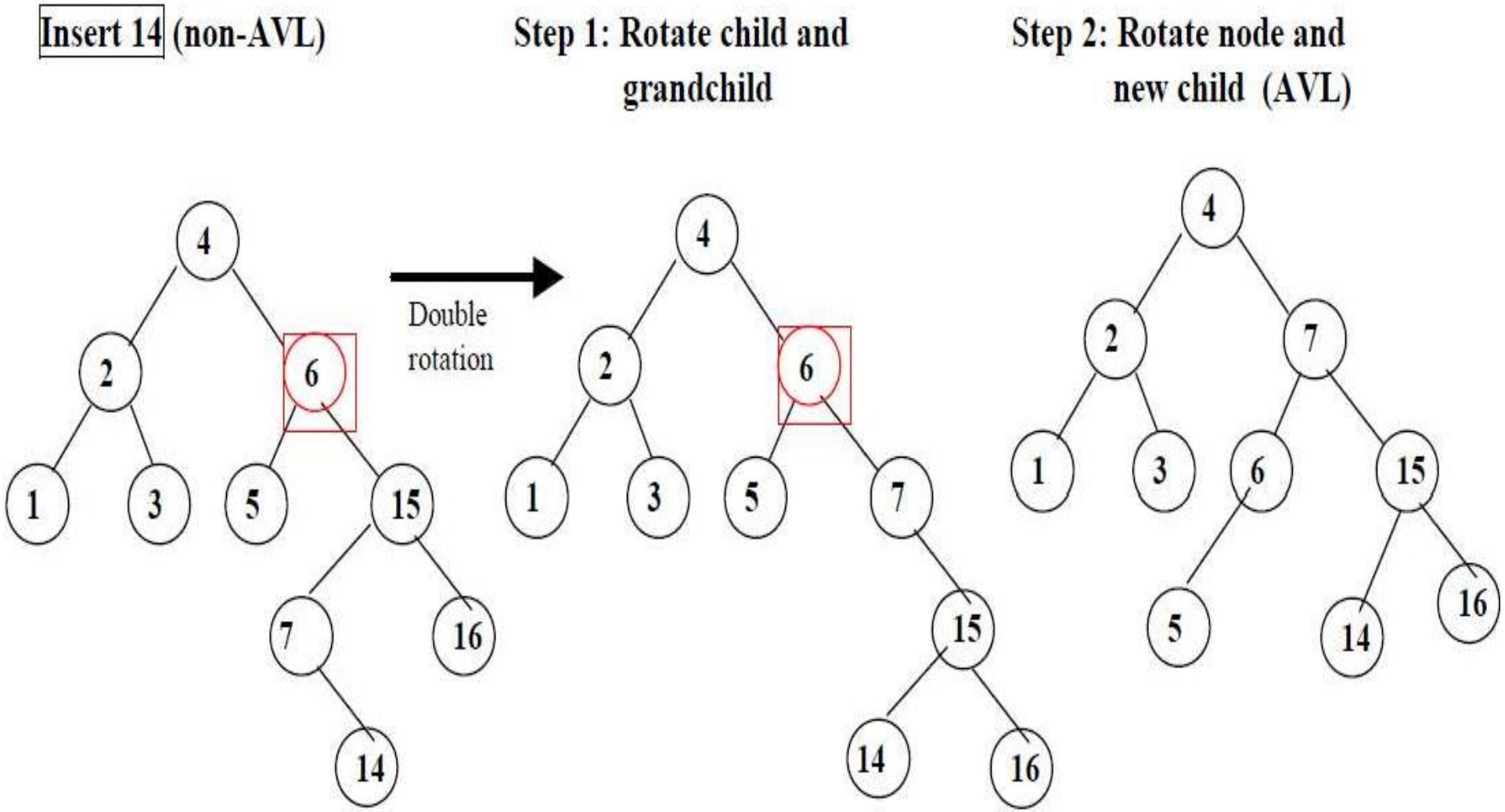
Step 1: Rotate child and grandchild

Step 2: Rotate node and new child (AVL





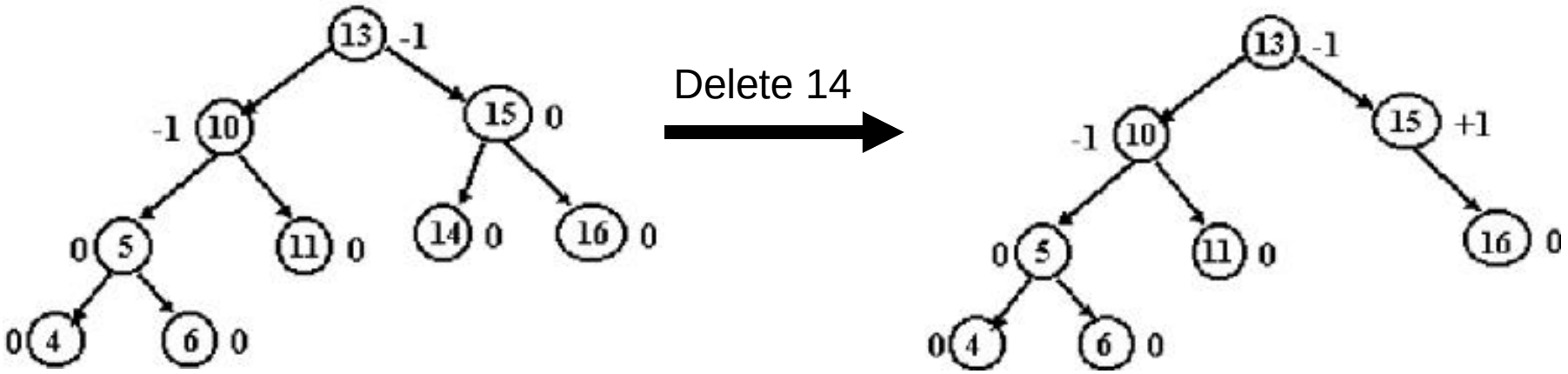
AVL Trees Example



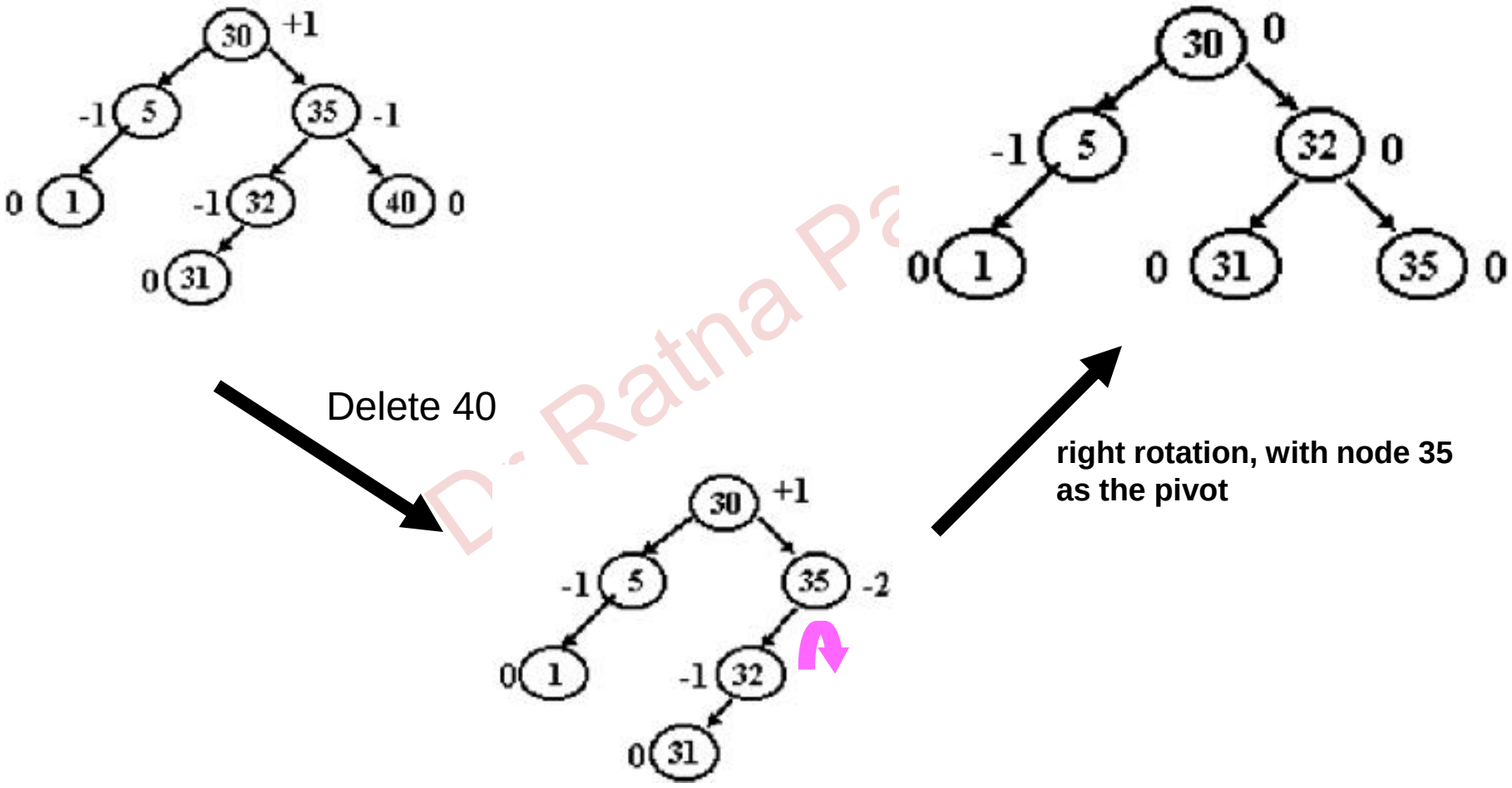
Deletion

- Delete by a BST deletion
- Rebalance the tree if an imbalance occurs.
- There are three deletion cases:
  1. Deletion that does not cause an imbalance.
  2. Deletion that requires a single rotation to rebalance.
  3. Deletion that requires two or more rotations to rebalance.
- Deletion case 1 example:

Dr Ratna Patil

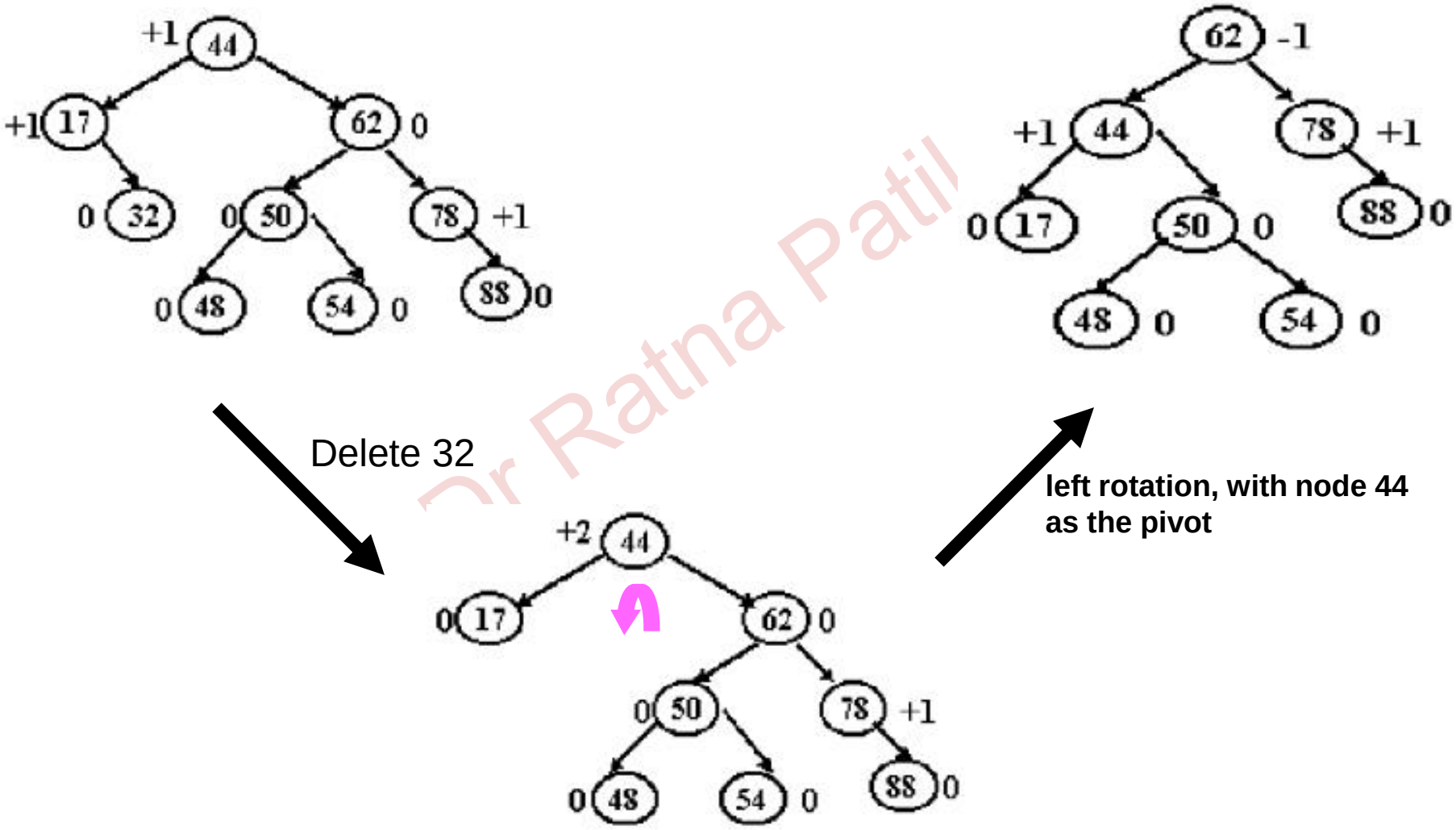


Deletion: case 2 examples



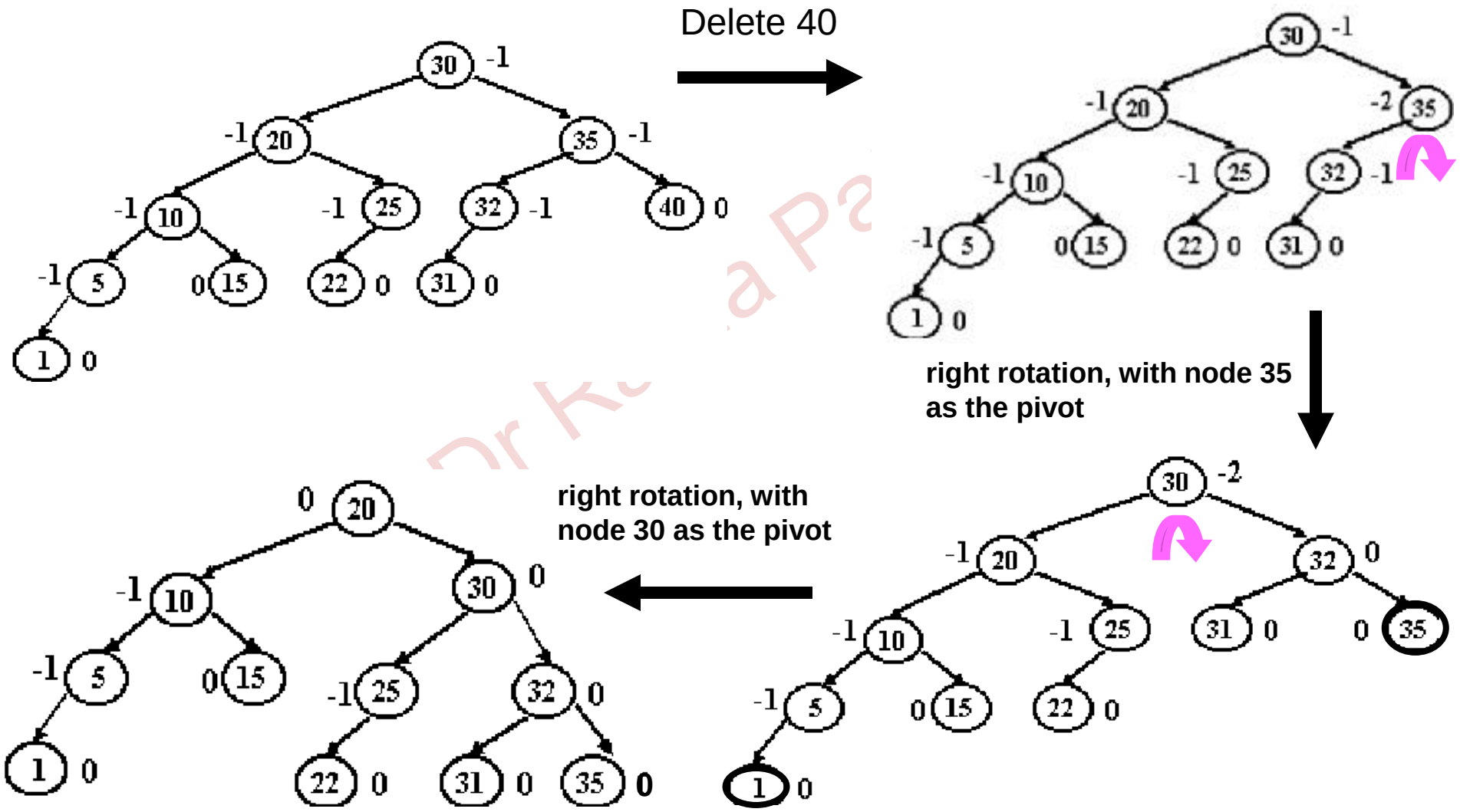
Dr. Ratna Pa

Deletion: case 2 examples (contd.)



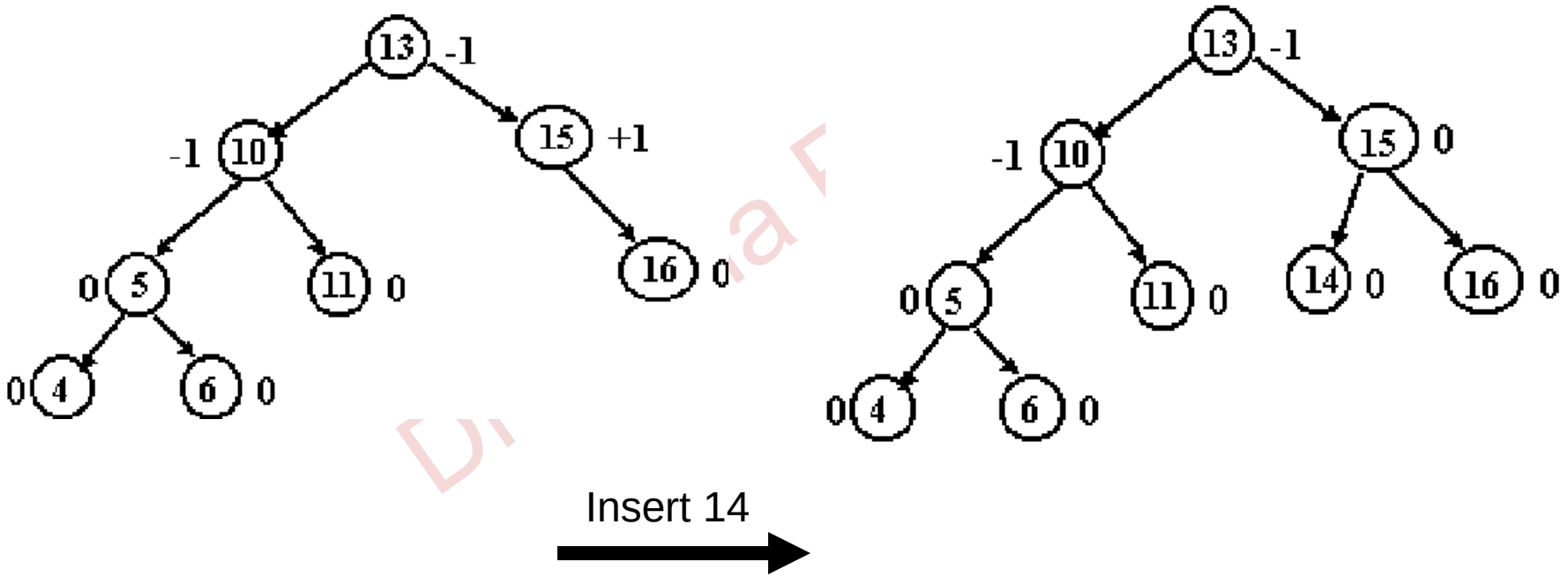
Dr Ratna Patil

Deletion: case 3 examples



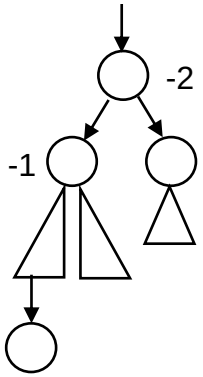
Insertion: case 1

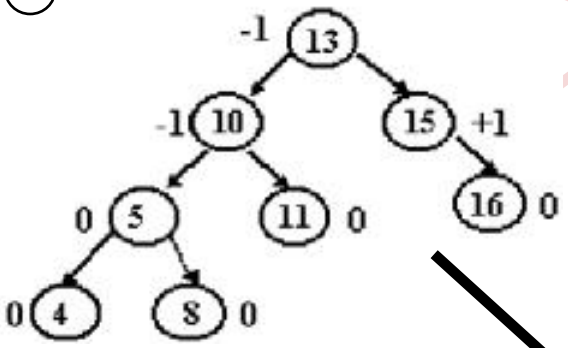
- Example: An insertion that does not cause an imbalance.



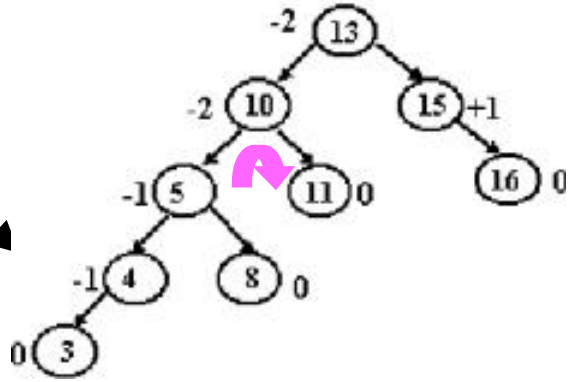
Insertion: case 2

- Case 2a: The lowest node (with a balance factor of -2) had a taller left-subtree and the insertion was on the left-subtree of its left child.
- Requires single right rotation to rebalance.

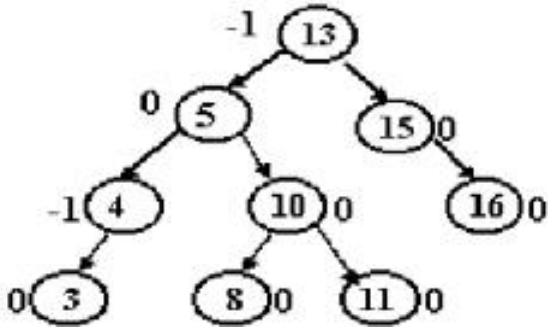




Insert 3



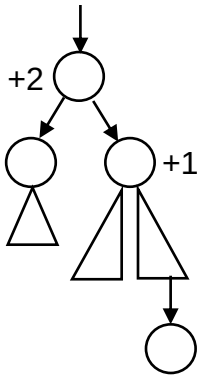
right rotation, with node 10 as pivot



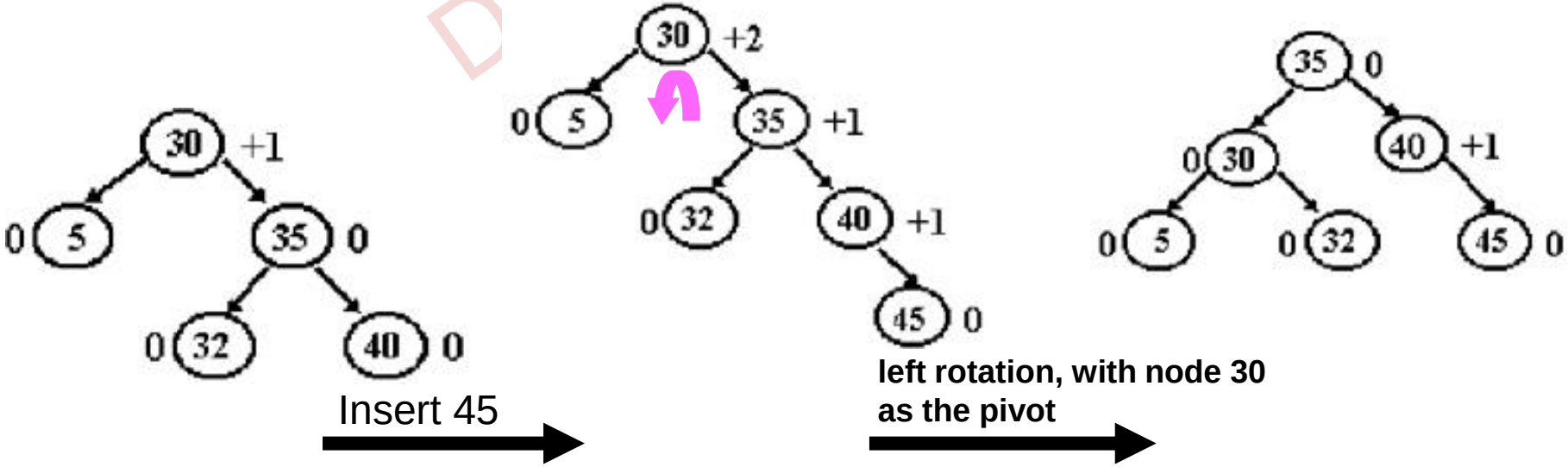
Dr Ratna Patil

Insertion: case 2 (contd.)

- Case 2b: The lowest node (with a balance factor of +2) had a taller right-subtree and the insertion was on the right-subtree of its right child.
- Requires single left rotation to rebalance.

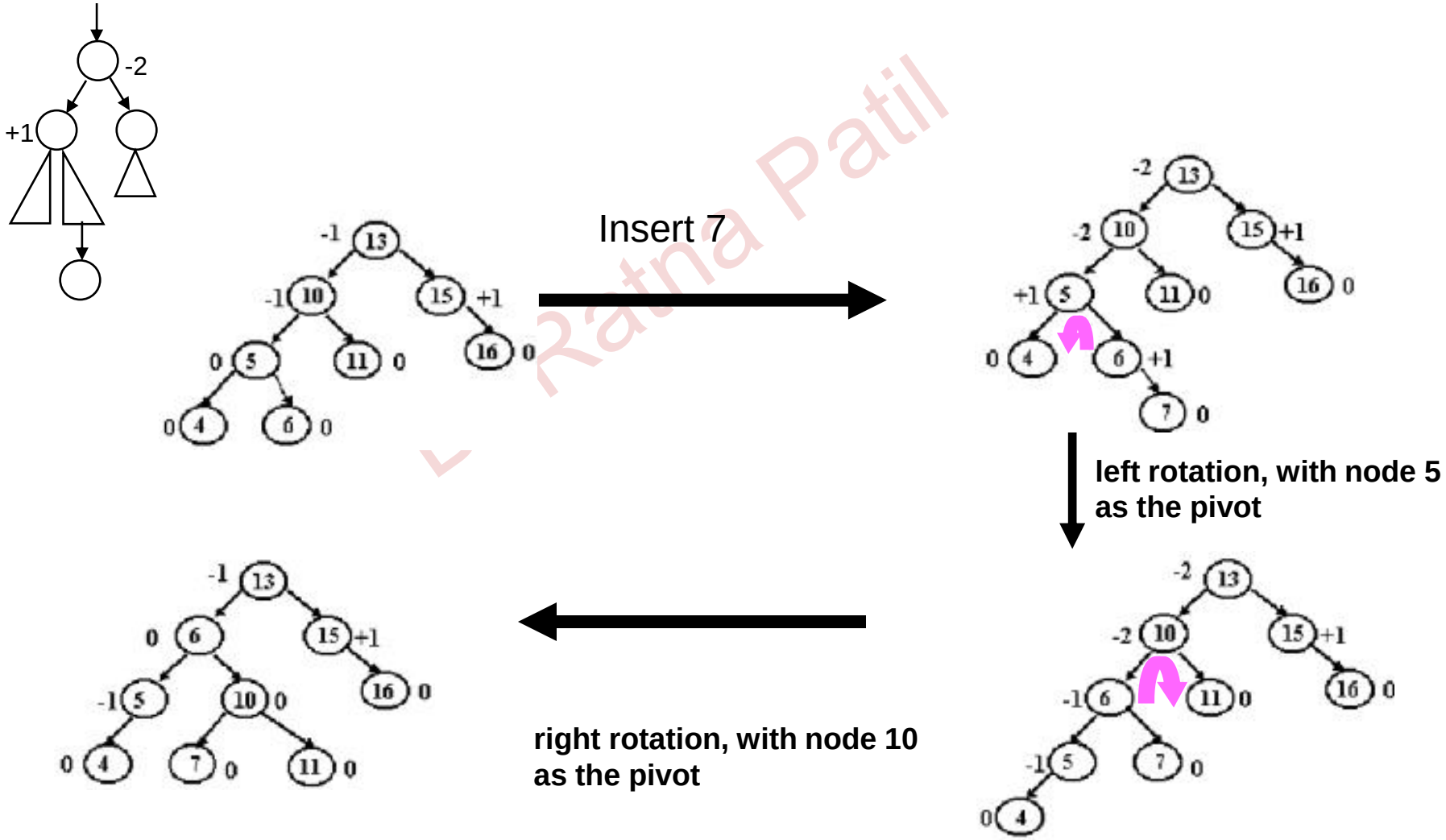


Dr Ratna Patil



Insertion: case 3

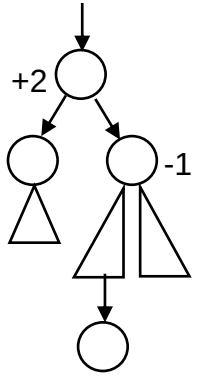
- Case 3a: The lowest node (with a balance factor of -2) had a taller left-subtree and the insertion was on the right-subtree of its left child.
- Requires a double left-right rotation to rebalance.

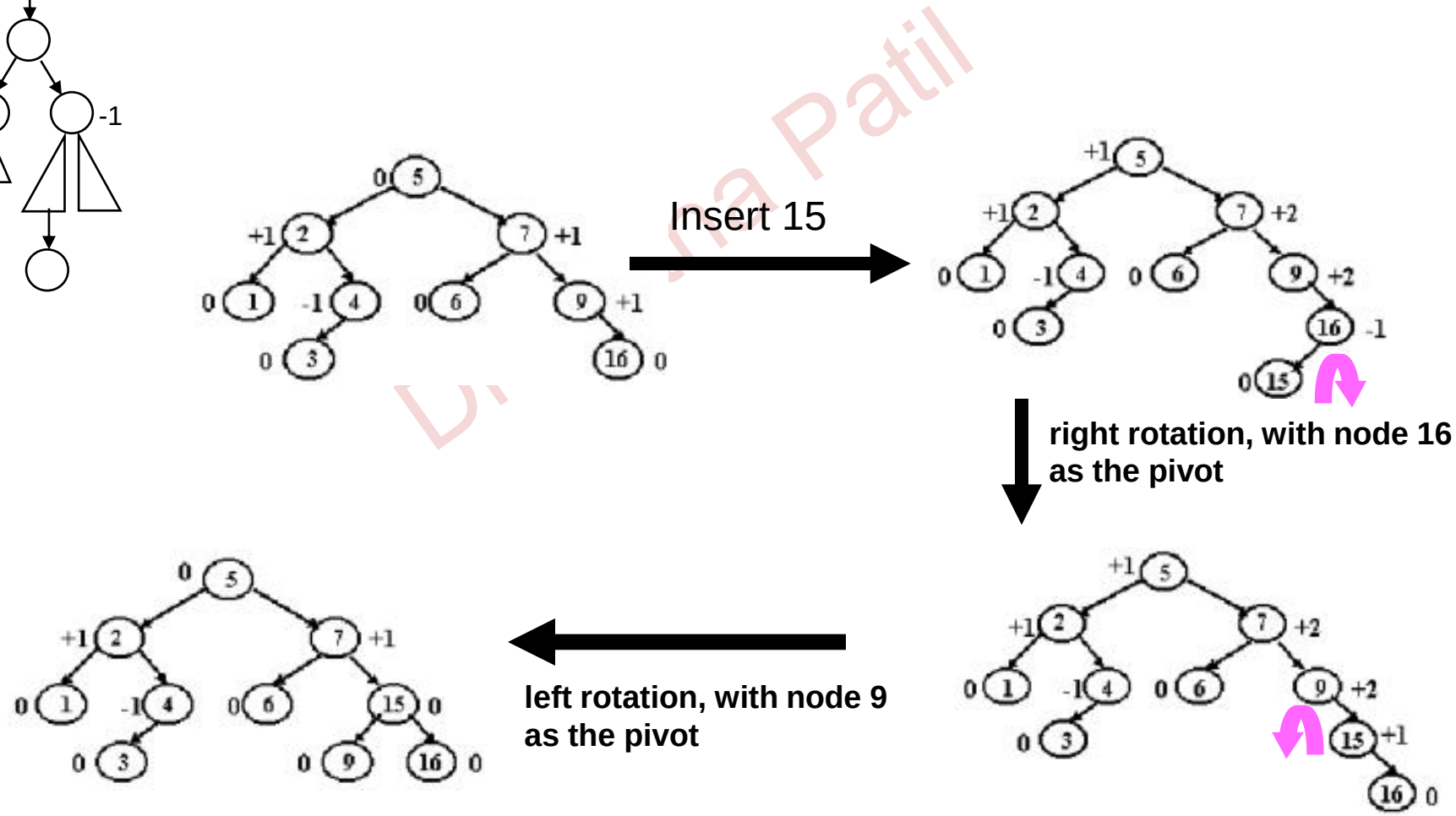


Ratna Patil

Insertion: case 3 (contd.)

- Case 3b: The lowest node (with a balance factor of +2) had a taller right-subtree and the insertion was on the left-subtree of its right child.
- Requires a double right-left rotation to rebalance.

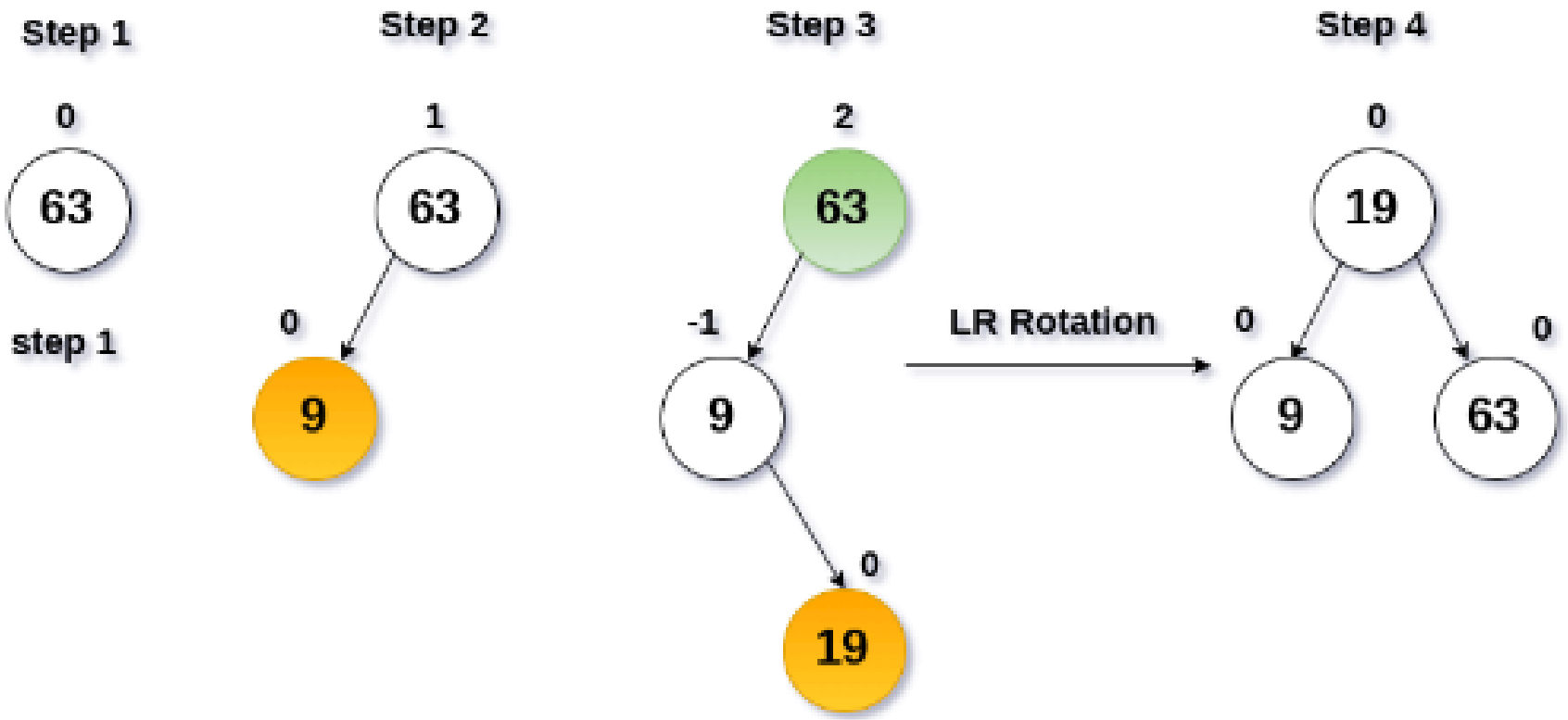




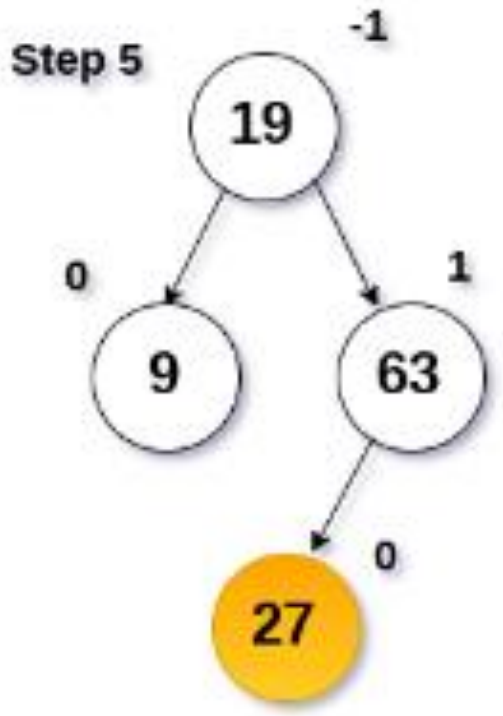
Dr. Jaya Patil

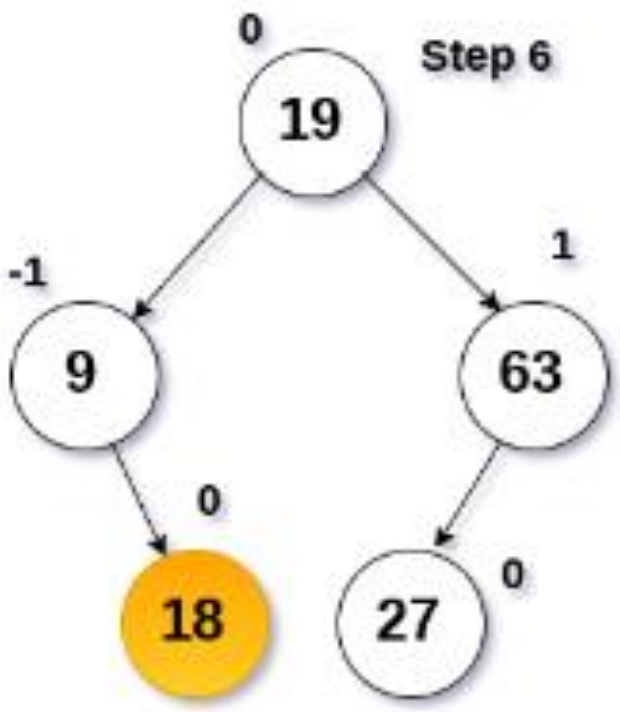
- Q. Construct an AVL tree by inserting the following elements in the given order.
- **63, 9, 19, 27, 18, 108, 99, 81**

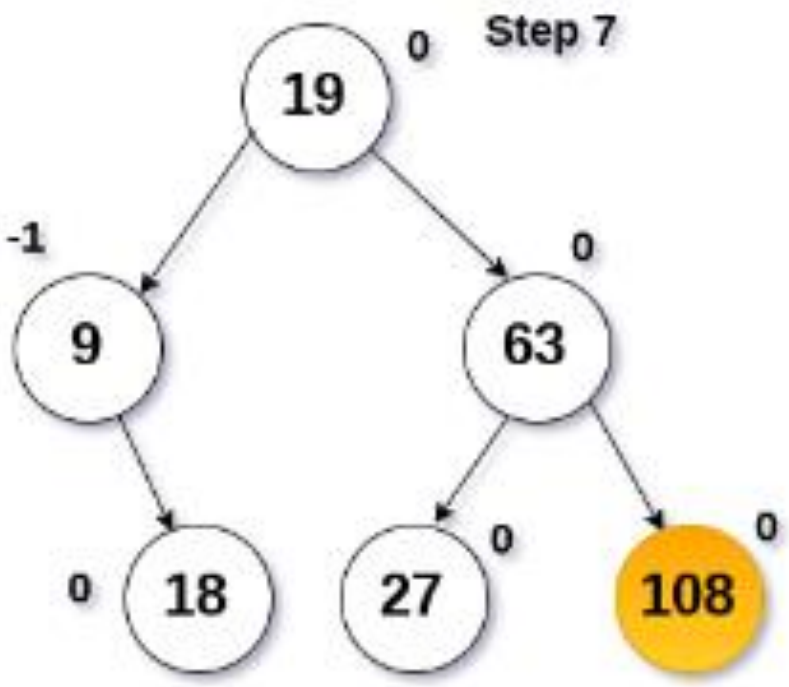
All the elements are inserted in order to maintain the order of binary search tree.



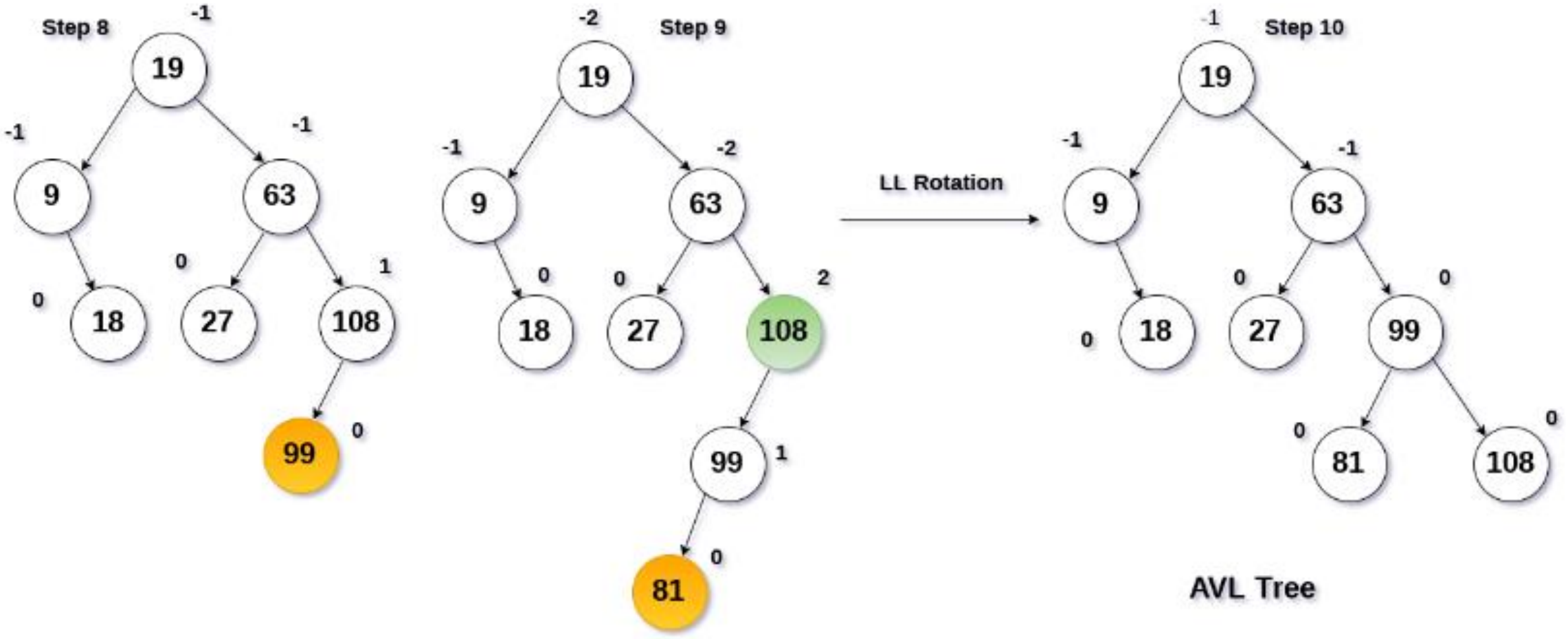
- Q. Construct an AVL tree by inserting the following elements in the given order.
- **63, 9, 19, 27, 18, 108, 99, 81**





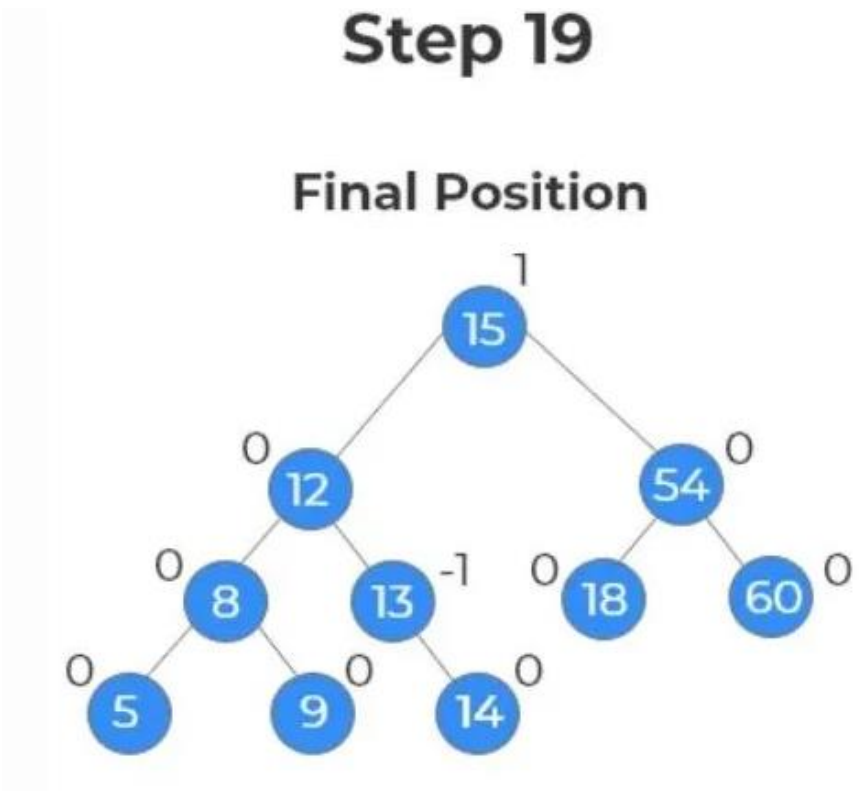


- Q. Construct an AVL tree by inserting the following elements in the given order.
- **63, 9, 19, 27, 18, 108, 99, 81**



- Q. Construct an AVL tree by inserting the following elements in the given order.

[15 , 18 , 12 , 8 , 54 , 5 , 14 , 13 , 9 , 60].



Patil

Important Questions

By SPPU Exam Pattern

1. Define Balance Factor of a node in a Binary Tree. 2M
2. What are the benefits of AVL Tree over BST? 4M
3. Explain Insertion operation in AVL tree with suitable example. 5M
4. Obtain AVL tree of following data element
   50,55,60,15,10,40,20,45,30,70,80. 8M
5. Build AVL Tree of Sun, Fri, Mon, Wed, Tue, Thurs, Sat. 8M
6. Build AVL Tree of A,Z,B,Y,C,X,D,U. 8M
7. Build AVL Tree of MAR,MAY,NOV,AUG,APR,JAN,DEC,JUL. 8M
   (Use lexicographic order i.e April ……. Sept)
8. Difference Between Red Black Tree & AVL Tree. 6M

Dr Ratna Patil

1. Construct AVL Tree for the following numbers

   14, 8, 12, 46, 23, 5, 77, 88, 20

2. Construct an AVL tree with the values 3,1,4,5,9,2,8,7,0 into an initially empty tree. Write code for inserting into AVL tree.

3. Write a routine for AVL tree insertion. Insert the following elements in the empty tree and how do you balance the tree after each element insertion Elements: 2, 5, 4, 6, 7, 9, 8, 3, 1, 10

Dr Ratna Patil

Multiway Trees

B-Trees and B+ Trees

Multiway Trees

- We now look at trees that are not restricted to one entry per node

- Used for internal and external indices and search trees, among other things

Dr Ratna Patil

Multiway search trees
Basic Theory

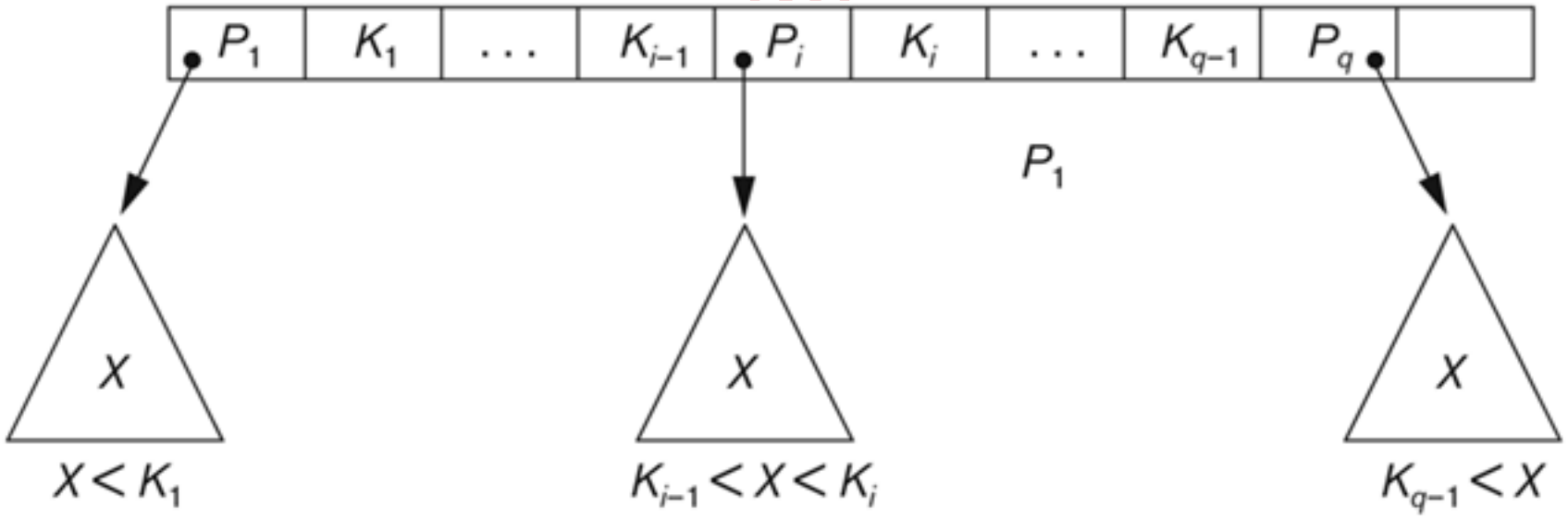
- “A multi-way search tree of order n is a general tree in which each node refers to n or few sub-trees and contains one few keys than it has sub-trees.”

Ratna Patil

Pointer to Keys < Key1	**Key 1**	Pointer to keys between Key1 and Key2	**Key 2**	Pointer to keys between Key2 and Key3	**Key 3**	Pointer to Keys > Key3

Multiway search trees
Basic Theory

A node in a serach tree with pointers to sub-trees below it



Say That Again, Please…

- Each node has *n* children and *n-1* keys
- The keys in each node are in ascending order
- The keys in the first *i* children are smaller than the ith key
- The keys in the last *n-i* children are larger than the ith key

Dr Ratna Patil

Multiway Search Tree

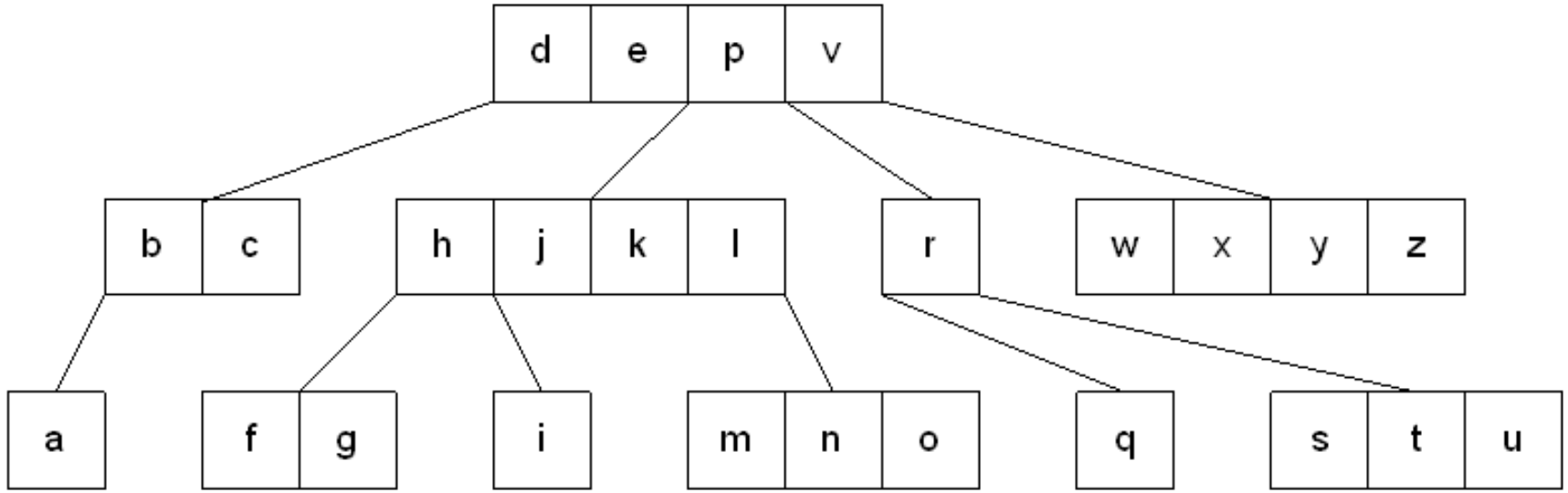
- A binary search tree has *one* value in each node and *two* subtrees.

- A **multiway tree** is a tree that can have more than two children. A **multiway tree of order m** (or an **m-way tree**) is one in which a tree can have m children.

- M is the order of the tree.

Dr Ratna Patil

Multiway tree of order 5



M way tree

- M way tree used to reduce the height of the tree
- M way tree is not balanced tree.
- An extension of a multiway search tree of order m is a **B-tree of order m.**

Dr. Ratna Patil

B-Trees

- Reduces the time required for accessing secondary storage

- The nodes can become very large - typically the size of a block

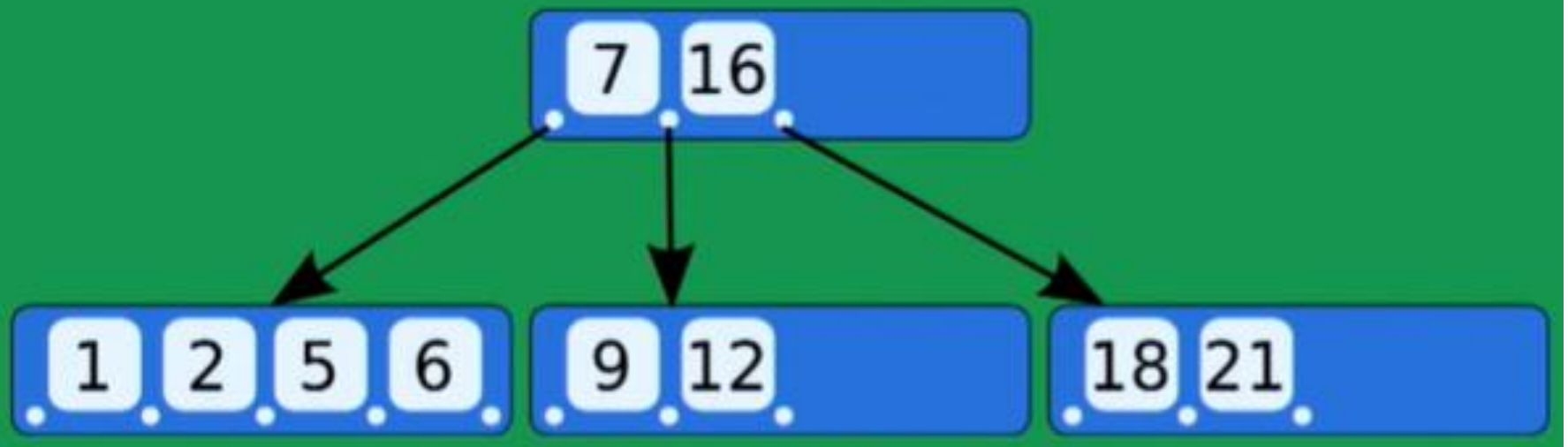
Dr Rathna Patil

B-Trees

A **B-tree** of order *n* is a multi-way search tree with the following properties:

- The root has at least one key.
- Each non-root node holds (k-1) keys and k pointers to sub-trees where: $\lceil n/2 \rceil \leq k \leq n$ (ceiling function)
- All leaves (terminal nodes) are at the same level.

Dr Ratna Patil



A B-Tree of Order M:

- Every node has at most *m* children
- A non-leaf node with *k* children contains *k*−1 keys
- The root has at least two children if it is not a leaf node
- Every non-leaf node (except root) has at least ⌈*m*/2⌉ children
- All leaves appear in the same level

B-Trees

According to these conditions, a B-Tree is

- always at *least half full*,
- has relatively few levels, and
- ‘perfectly’ balanced.

Dr Ratna Patil

B-Trees

B_3-Tree (non-root nodes at full and lowest capacity)

ptr1	**key1**	ptr2	**key2**	ptr3

ptr1	**key1**	ptr2

B_5-Tree (non-root nodes at full and lowest capacity)

ptr1	**key1**	ptr2	**key2**	ptr3	**key3**	ptr4	**key4**	ptr5

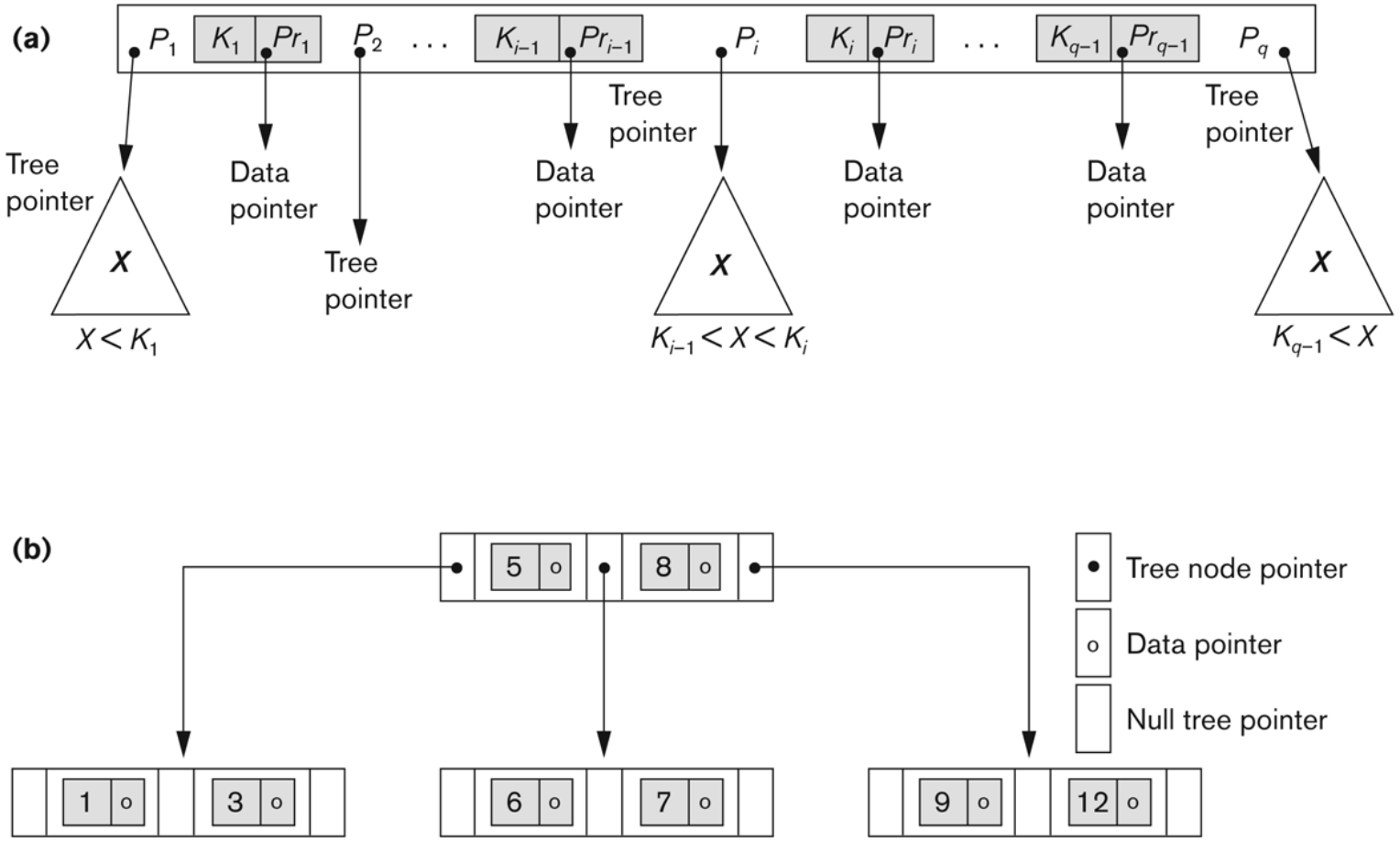
ptr1	**key1**	ptr2	**key2**	ptr3

Dr Ratna Patil

B-Trees

Figure 14.10

B-Tree structures. (a) A node in a B-tree with $q - 1$ search values. (b) A B-tree of order $p = 3$.The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.



B Tree

- B Tree is a specialized m-way tree that can be widely used for disk access. A B-Tree of order m can have at most m-1 keys and m children.
- One of the main reason of using B tree is its capability to store large number of keys in a single node and large key values by keeping the height of the tree relatively small.
- A B tree of order m contains all the properties of an M way tree. In addition, it contains the following properties.

1. Every node in a B-Tree contains at most m children.
2. Every node in a B-Tree except the root node and the leaf node contain at least m/2 children.
3. The root nodes must have at least 2 nodes.
4. All leaf nodes must be at the same level.

It is not necessary that, all the nodes contain the same number of children but, each node must have m/2 number of nodes.

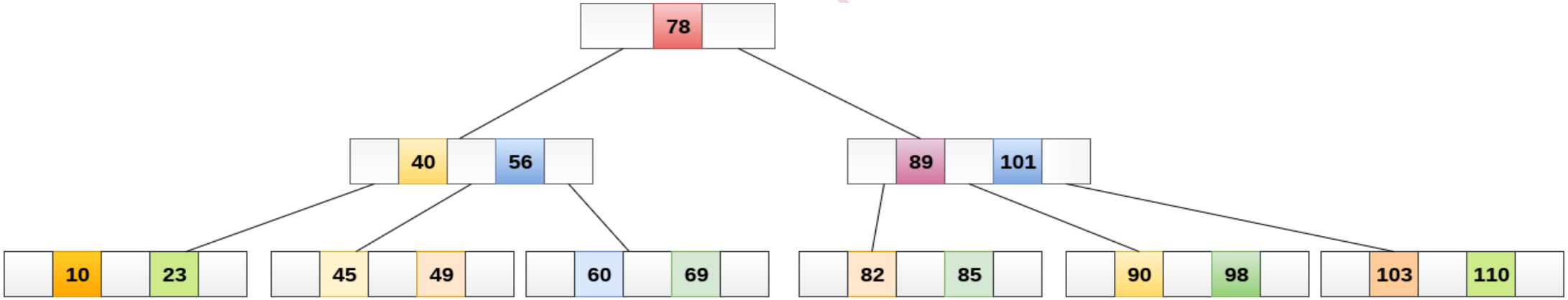
Dr Ratna Patil

Searching :

Searching in B Trees is similar to that in Binary search tree. For example, if we search for an item 49 in the following B Tree. The process will something like following :

1. Compare item 49 with root node 78. since 49 < 78 hence, move to its left sub-tree.
2. Since, 40<49<56, traverse right sub-tree of 40.
3. 49>45, move to right. Compare 49.
4. match found, return.

Patil



Inserting

Insertions are done at the leaf node level. The following algorithm needs to be followed in order to insert an item into B Tree.

1. Traverse the B Tree in order to find the appropriate leaf node at which the node can be inserted.
2. If the leaf node contain less than m-1 keys then insert the element in the increasing order.
3. Else, if the leaf node contains m-1 keys, then follow the following steps.

- Insert the new element in the increasing order of elements.
- Split the node into the two nodes at the median.
- Push the median element upto its parent node.
- If the parent node also contain m-1 number of keys, then split it too by following the same steps.

Dr Ratna Patil

Deletion

Deletion is also performed at the leaf nodes. The node which is to be deleted can either be a leaf node or an internal node. Following algorithm needs to be followed in order to delete a node from a B tree.

1. Locate the leaf node.
2. If there are more than m/2 keys in the leaf node then delete the desired key from the node.
3. If the leaf node doesn't contain m/2 keys then complete the keys by taking the element from eight or left sibling.

- If the left sibling contains more than m/2 elements then push its largest element up to its parent and move the intervening element down to the node where the key is deleted.
- If the right sibling contains more than m/2 elements then push its smallest element up to the parent and move intervening element down to the node where the key is deleted.

4. If neither of the sibling contain more than m/2 elements then create a new leaf node by joining two leaf nodes and the intervening element of the parent node.
5. If parent is left with less than m/2 nodes then, apply the above process on the parent too.

If the node which is to be deleted is an internal node, then replace the node with its in-order successor or predecessor. Since, successor or predecessor will always be on the leaf node hence, the process will be similar as the node is being deleted from the leaf node.

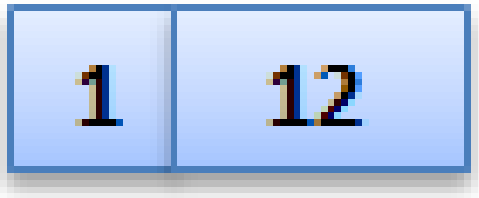
Dr. Ratna Patil

Insertion Example :-

- key :- 1,12,8,2,25,6,14,28,17,7,52,16,48,68.
  Order = 5
  Procedure for adding key in b-tree

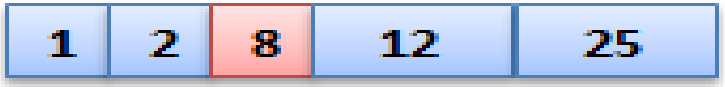
Step1.

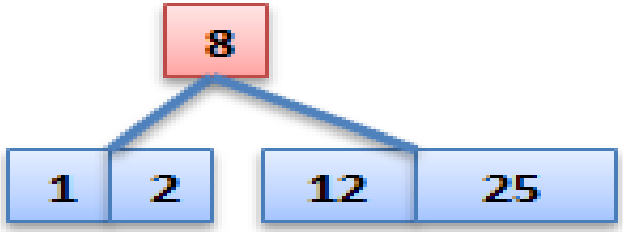
Step2.



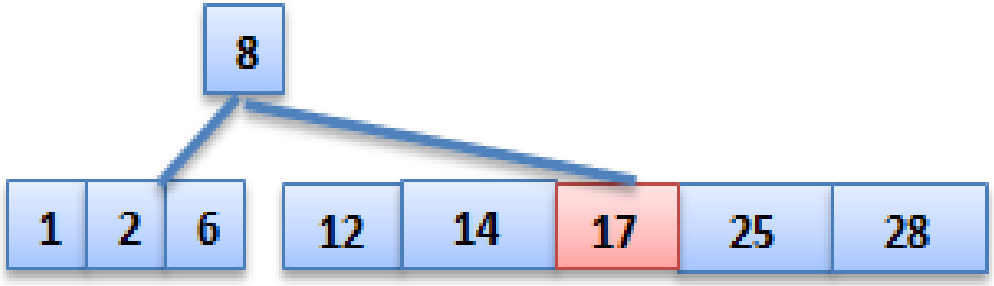
Dr Ratna Patil

Step3.

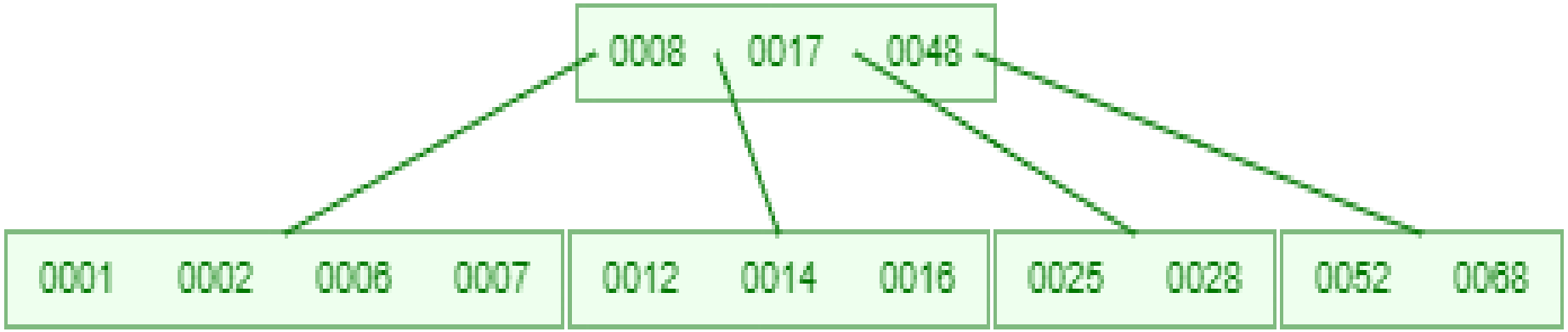




Step4.

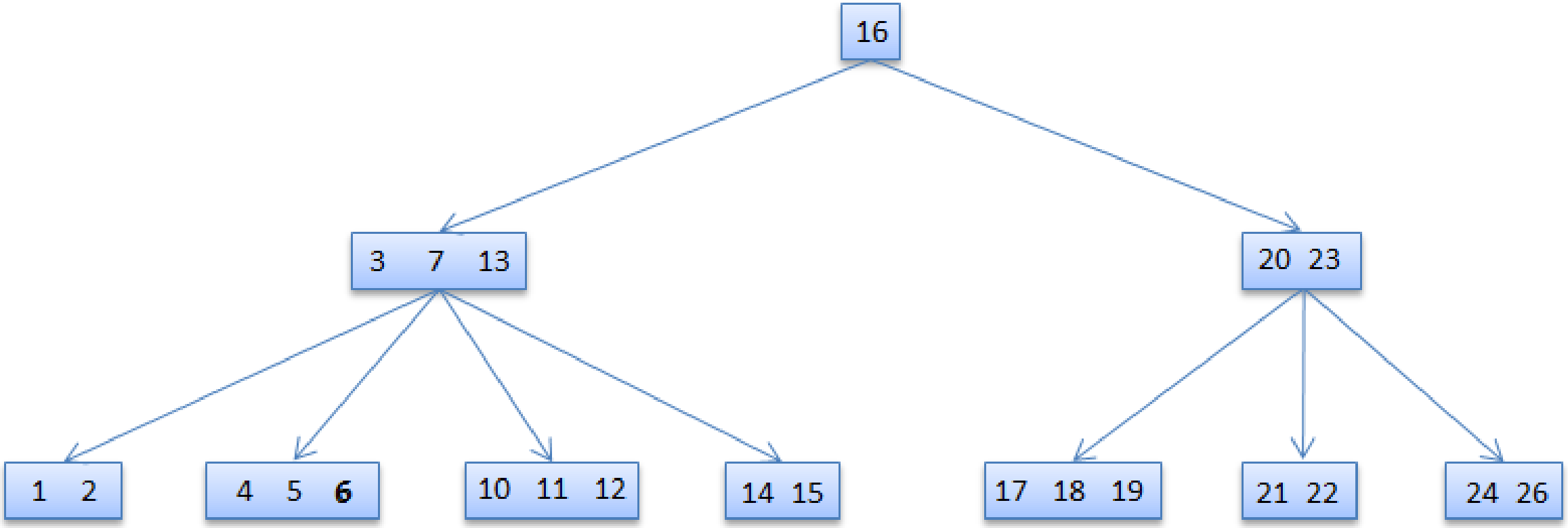


Step5.

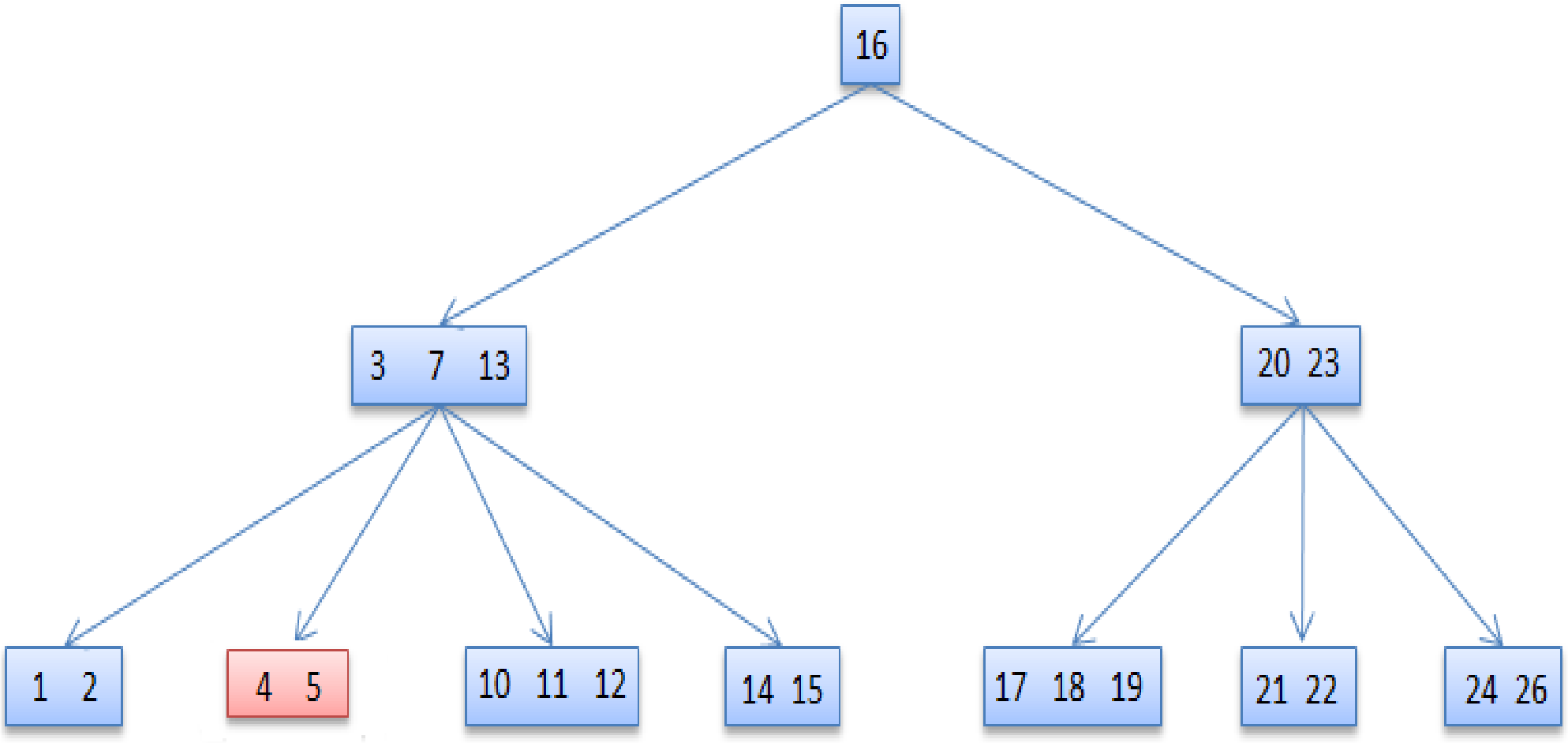


Deletion Example:-

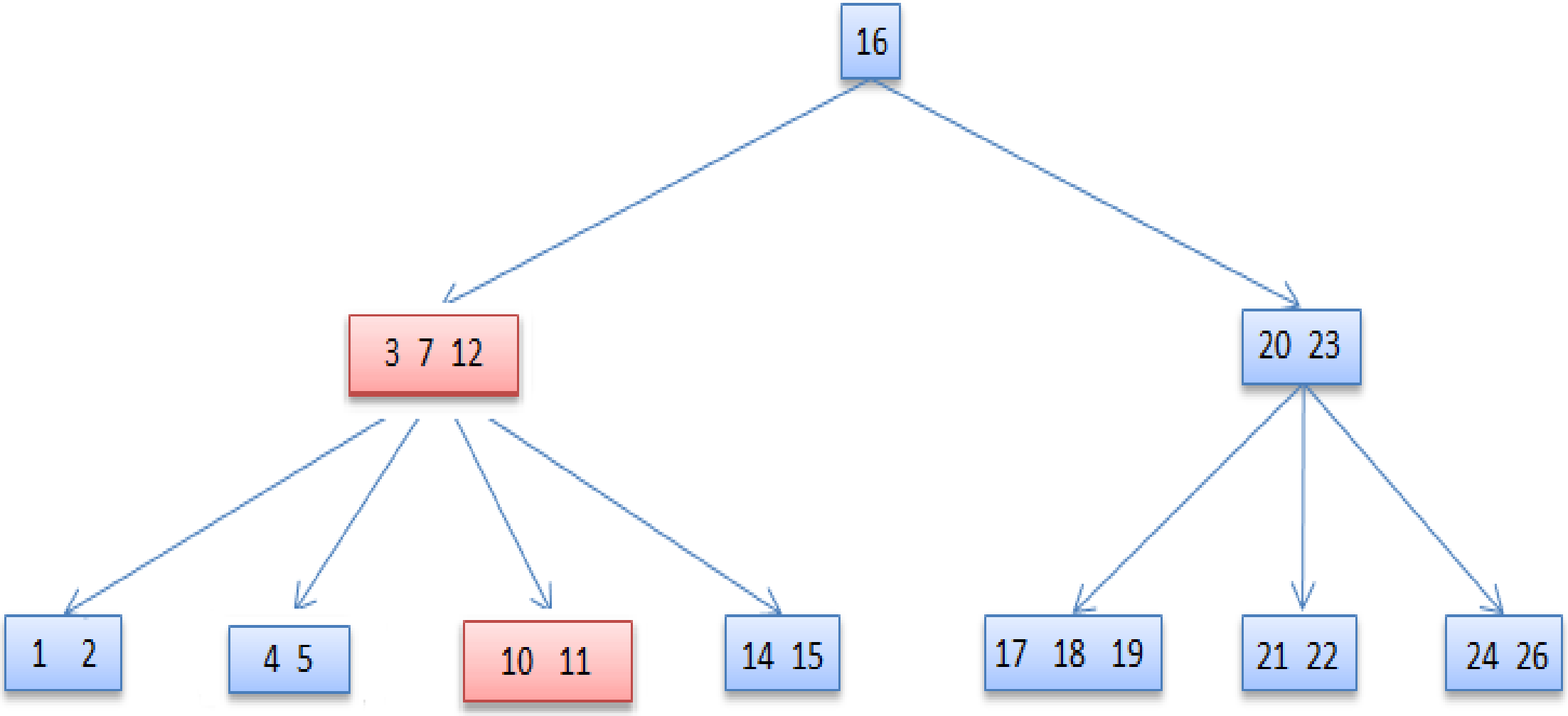
- **<u>Given Tree:-</u>**



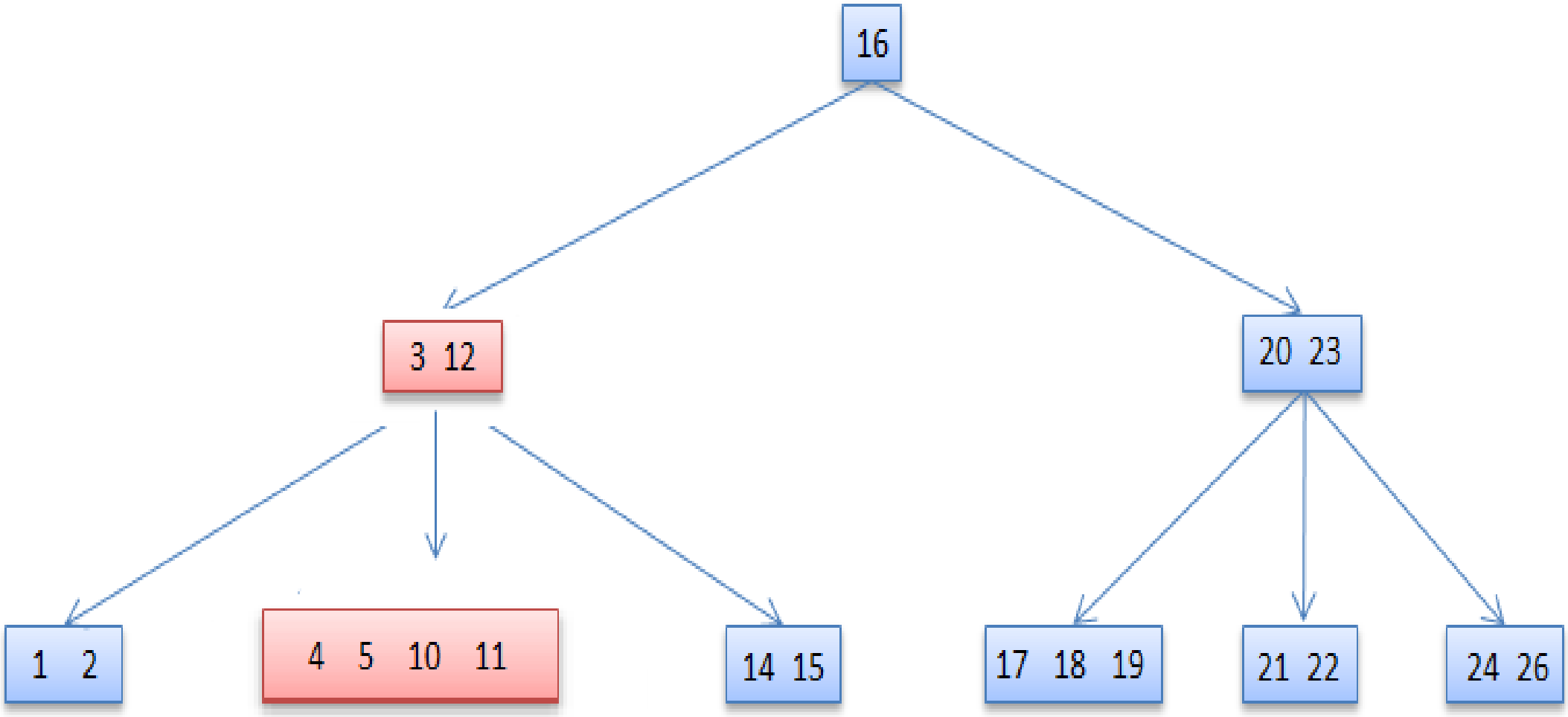
6 deleted



13 deleted



7 deleted



B^+ Trees

- Enhancement of B-Trees
- Allows all data to be accessed sequentially
- References to data are made only from leaves
- Internal nodes are *indexed* for faster access
- Leaves are indexed sequentially
- Leaves store keys, references to records in a data file associated with the keys, and a pointer to the next leaf

Dr Ratna Patil

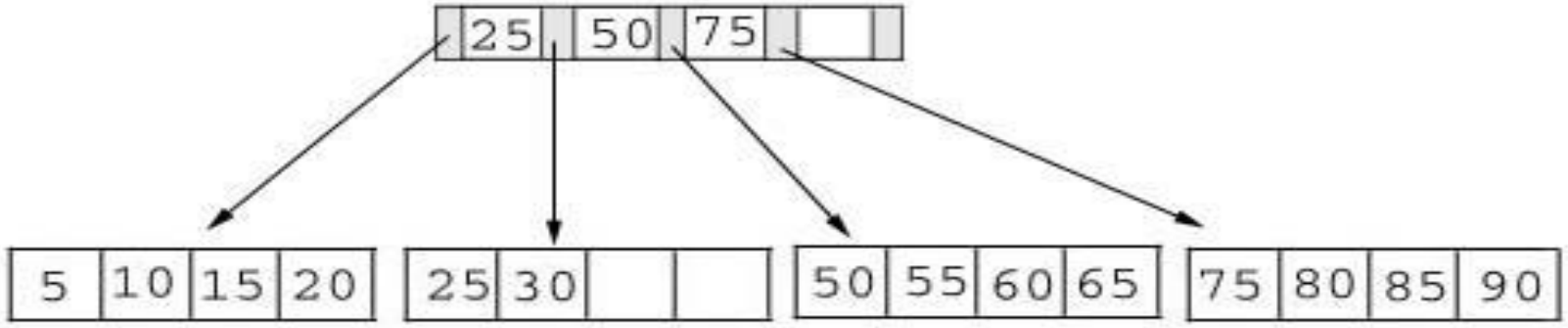
Insertion in B+ Tree

Step 1: Insert the new node as a leaf node
Step 2: If the leaf doesn't have required space, split the node and copy the middle node to the next index node.
Step 3: If the index node doesn't have required space, split the node and copy the middle element to the next index page.

Dr Ratna Patil

Insertion Example :-

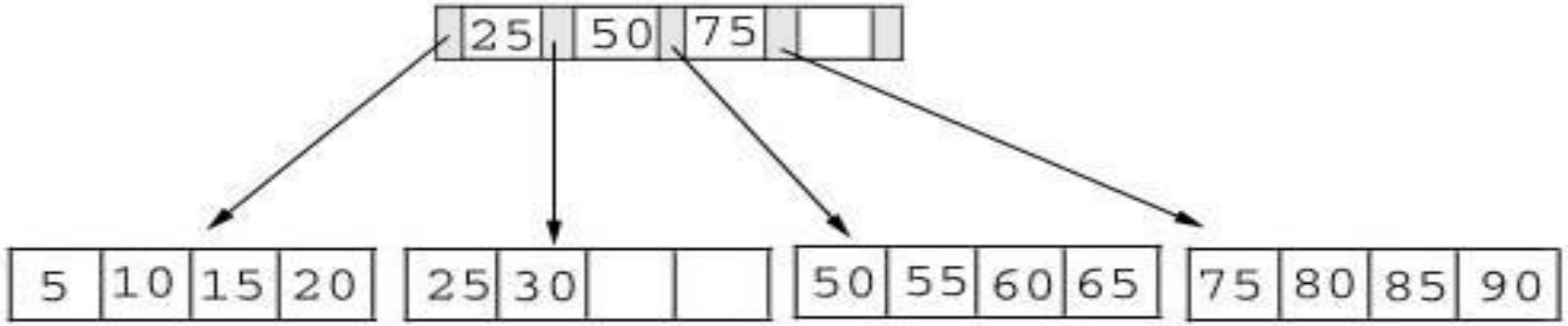
- Example #1: insert 28 into the below tree.



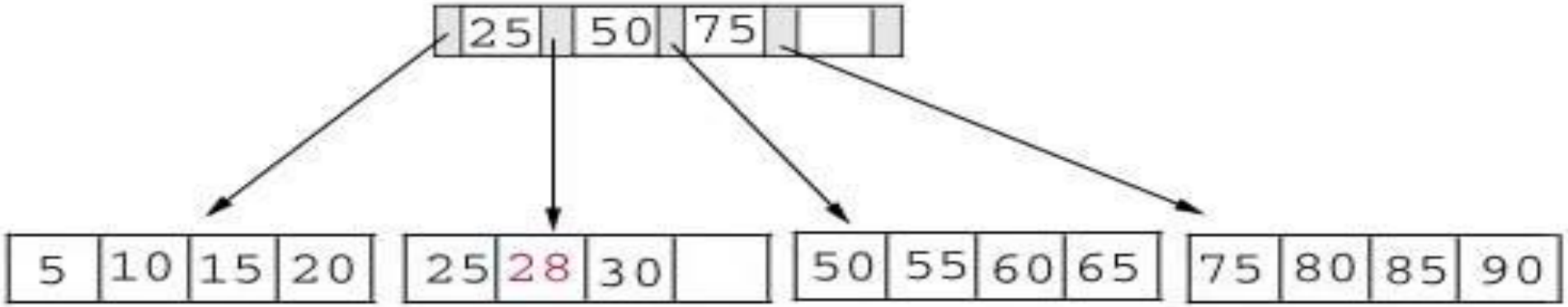
<u>Result:-</u>

Insertion Example :-

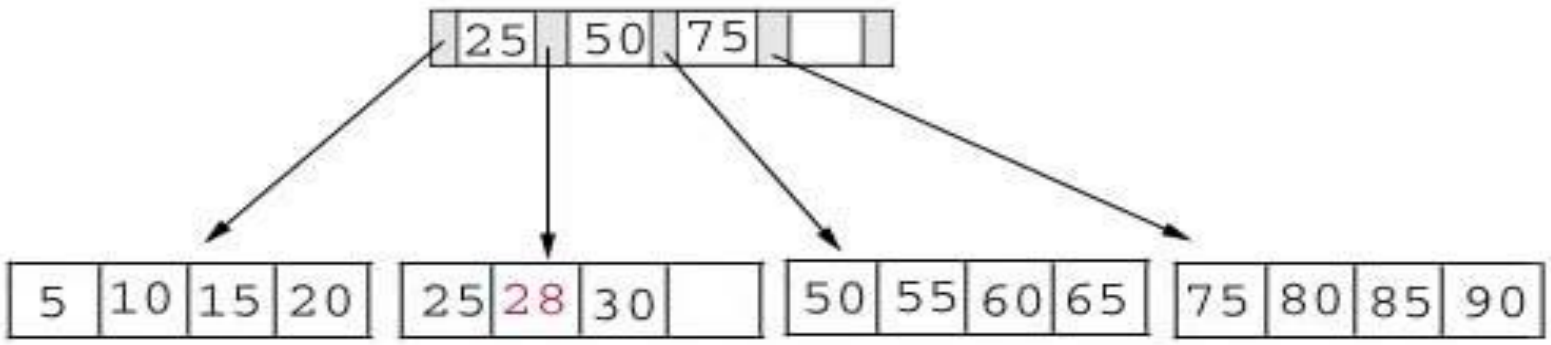
- Example #1: insert 28 into the below tree.



<u>Result:-</u>



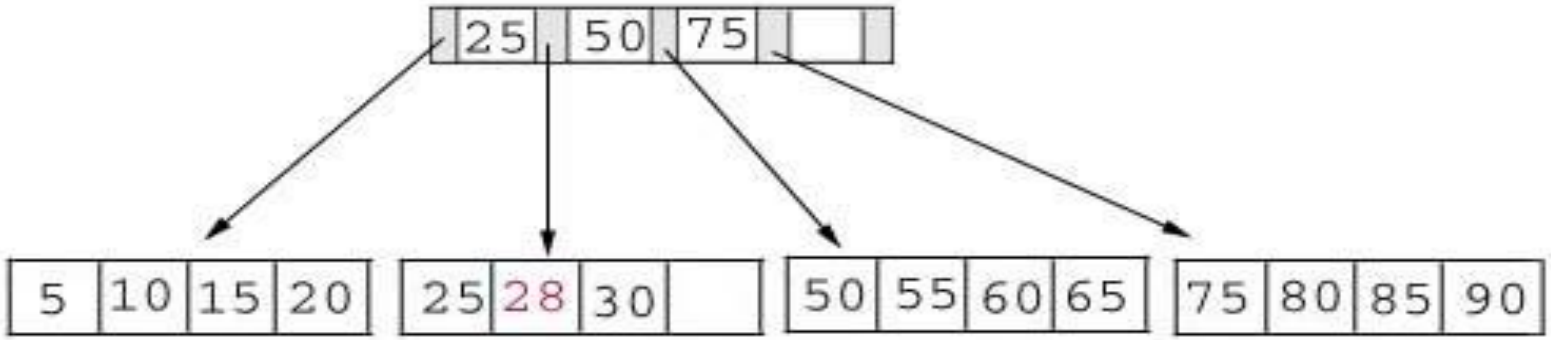
- Example #2: insert 70 into below tree



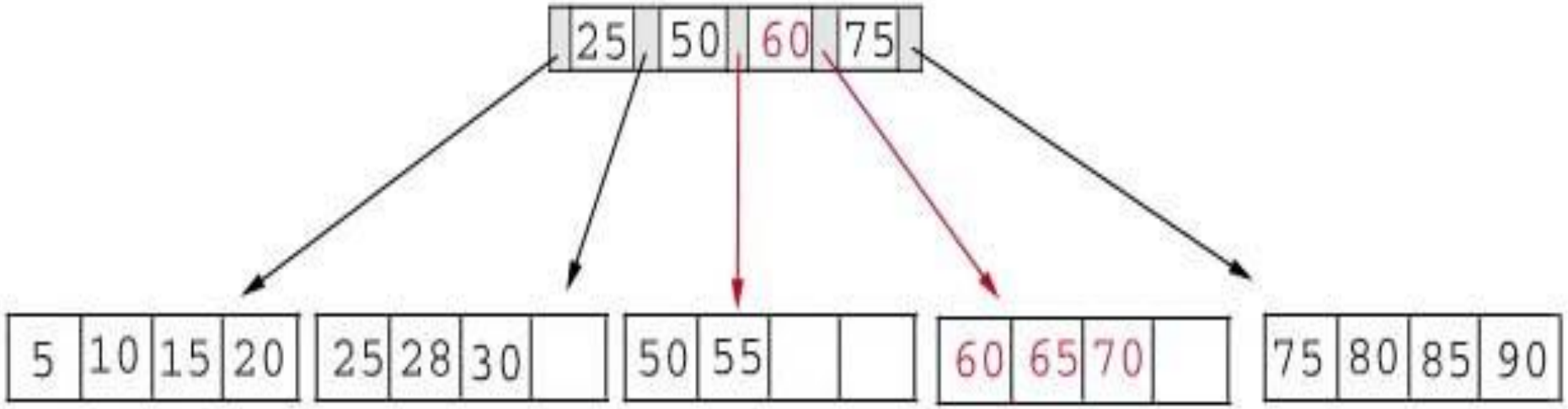
<u>Result:-</u>

Dr Rau

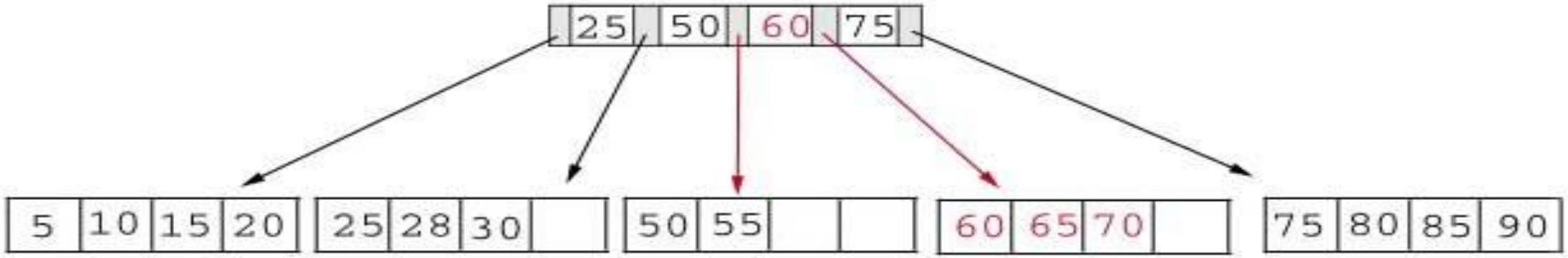
- Example #2: insert 70 into below tree



Result:-



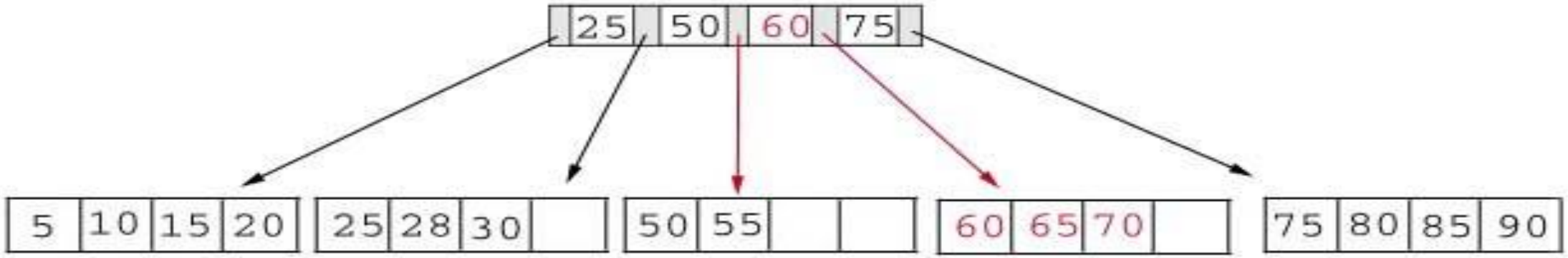
- Example #3: insert 95 into below tree



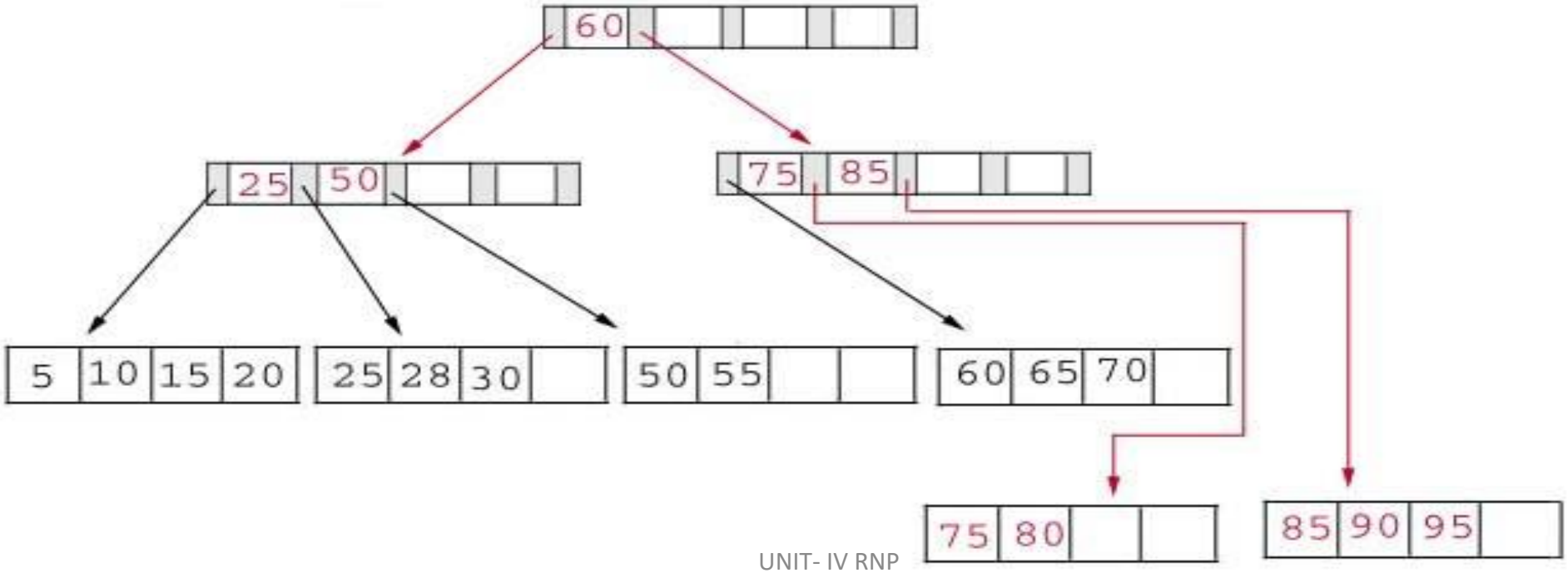
<u>Result:-</u>

Dr Ratna

- Example #3: insert 95 into below tree



Result:-



Deletion in B+ Tree

Step 1: Delete the key and data from the leaves.

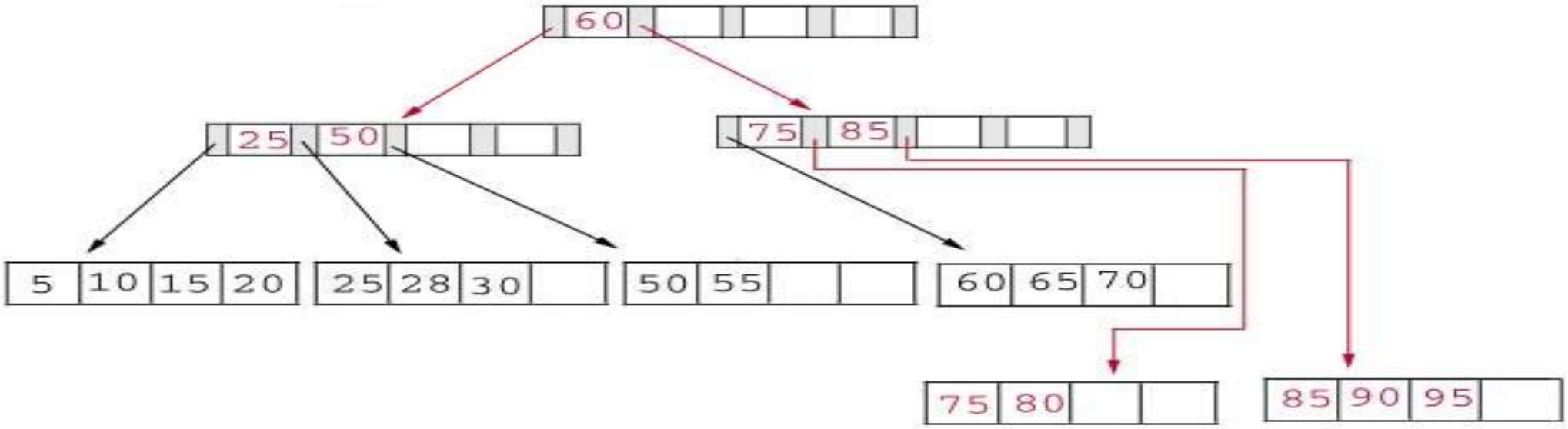
Step 2: if the leaf node contains less than minimum number of elements, merge down the node with its sibling and delete the key in between them.

Step 3: if the index node contains less than minimum number of elements, merge the node with the sibling and move down the key in between them.

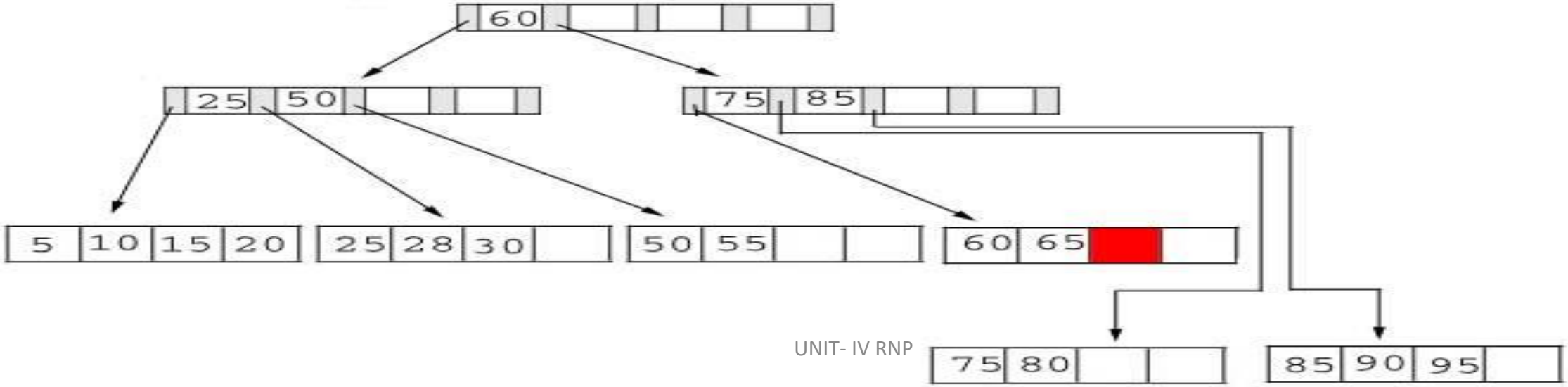
Dr Ratna Patil

Deletion Example :-

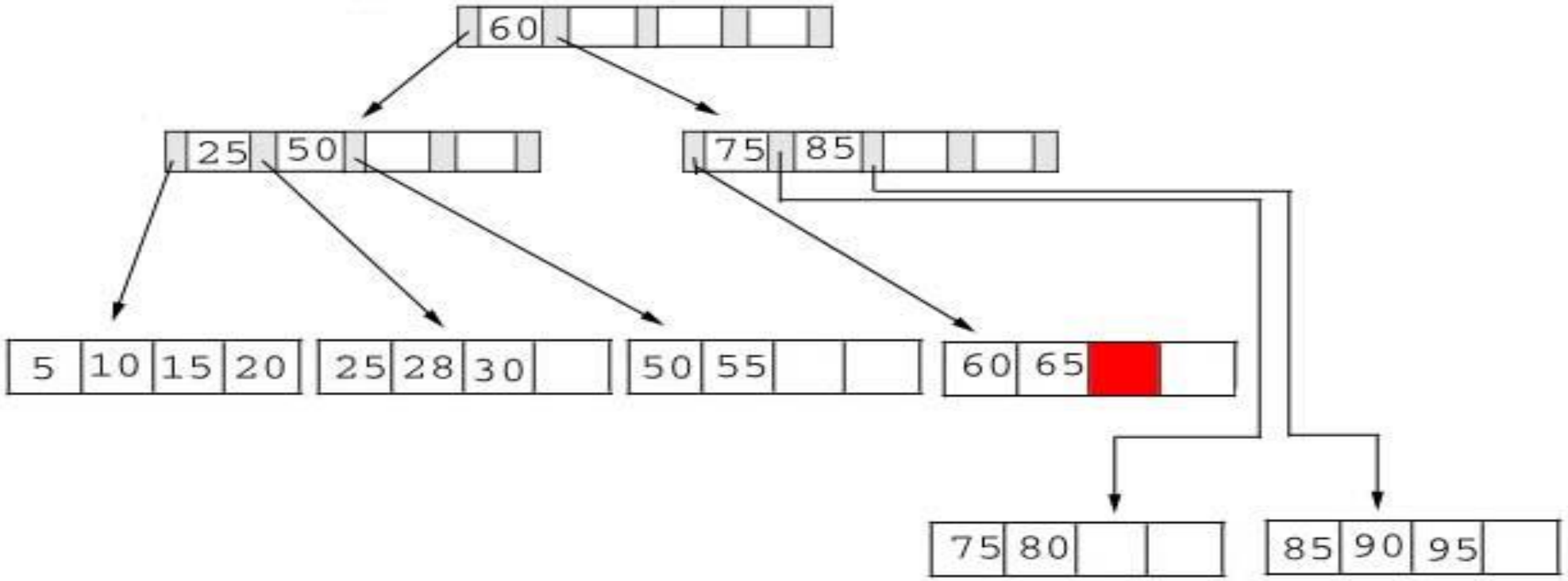
- Example #1: delete 70 from the tree



Result:-



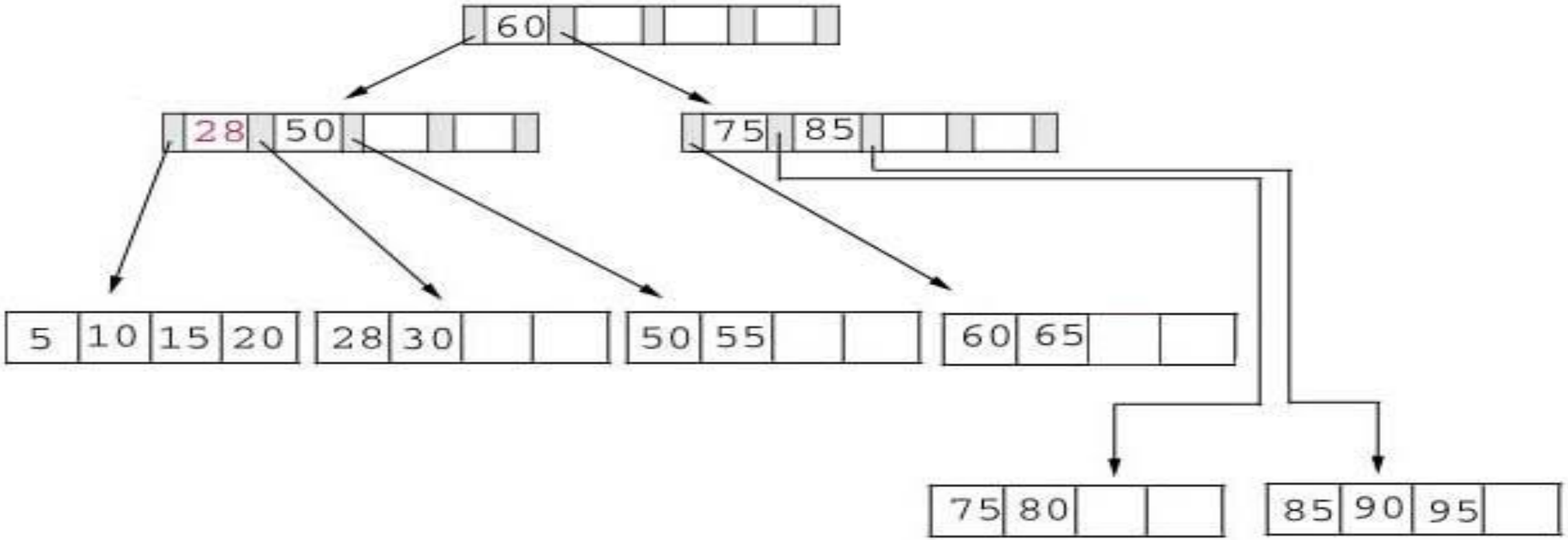
- Example #2: delete 25 from below tree



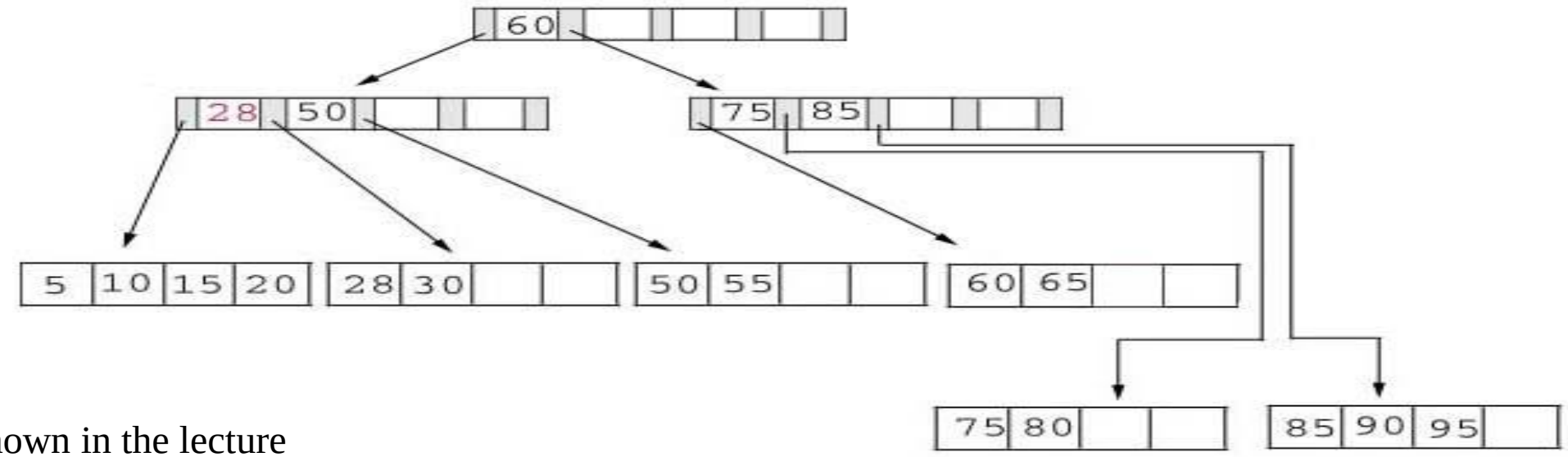
Result:-

Replace 25 with 28 in the index page.

On Next Slide…


60
28
50
75
85
5
10
15
20
28
30
50
55
60
65
75
80
85
90
95

- Example #3: delete 60 from the below tree



<u>Result:- Shown in the lecture</u>

Example of B+ tree

Show the B+ tree after insertions

- Suppose each B+ tree node can hold up to 4 pointers and 3 keys.
- Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

- Insert 1

Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

- Insert 3, 5

Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

1		

- Insert 3, 5

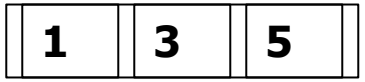
1	3	5

Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

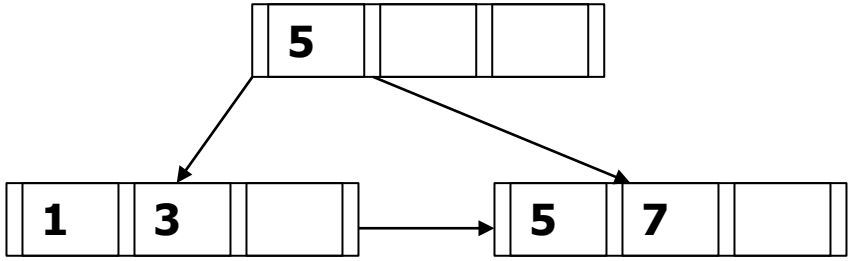
1	3	5

- Insert 7

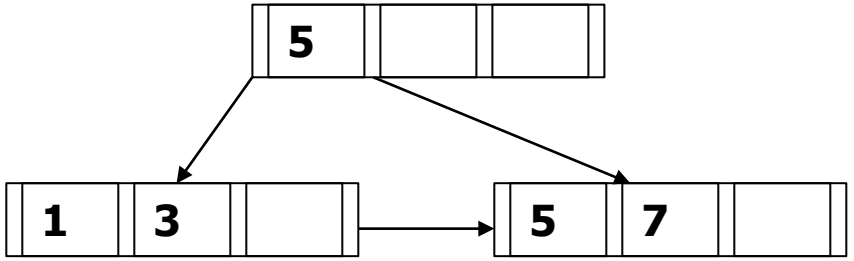
Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10



- Insert 7

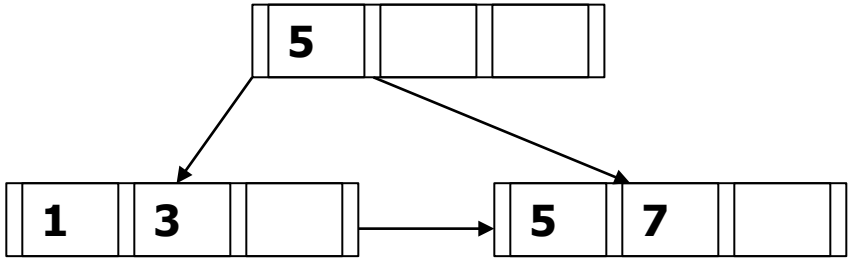


Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

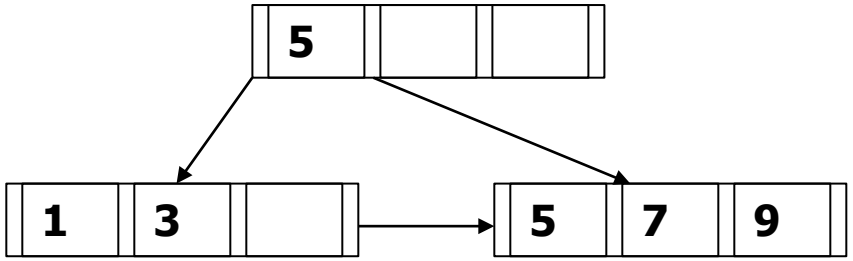


- Insert 9

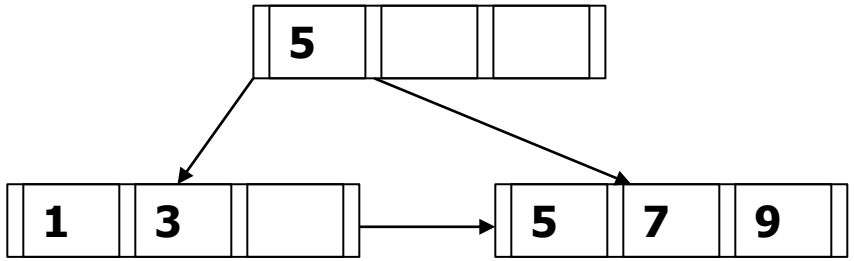
Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10



- Insert 9

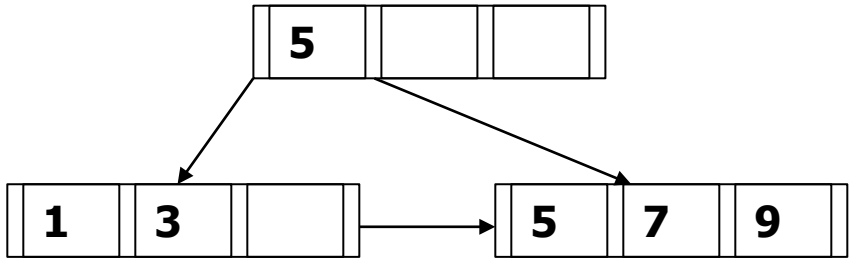


Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

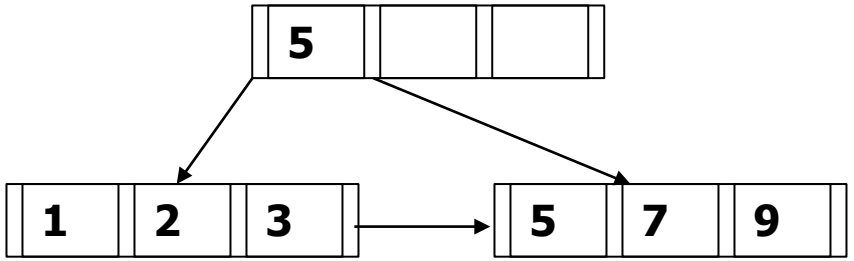


- Insert 2

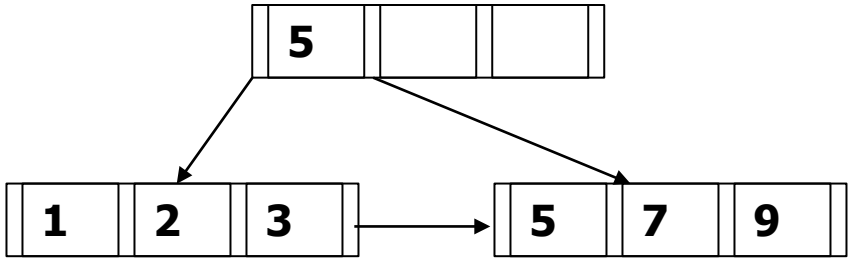
Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10



- Insert 2

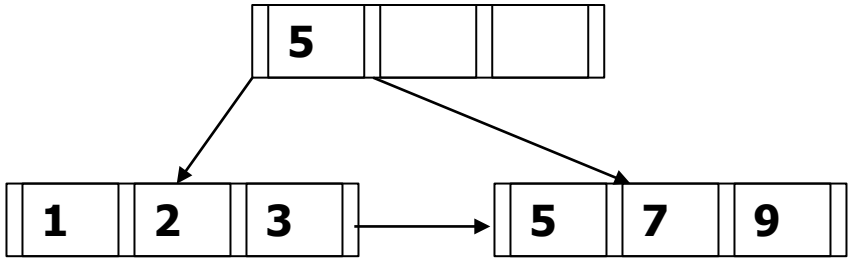


Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

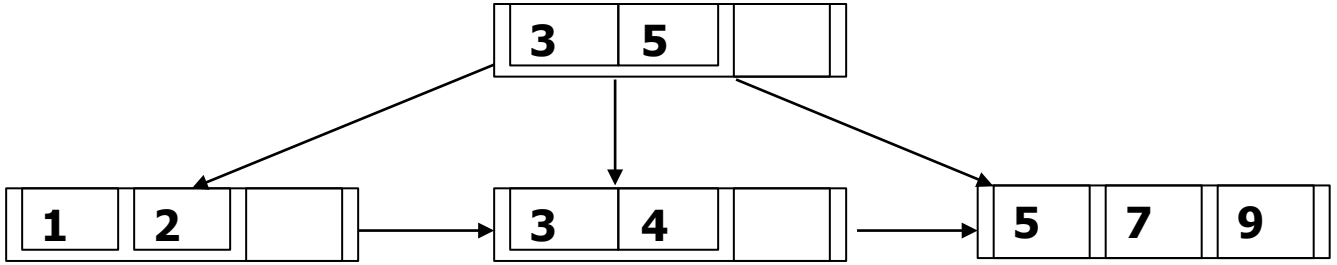


- Insert 4

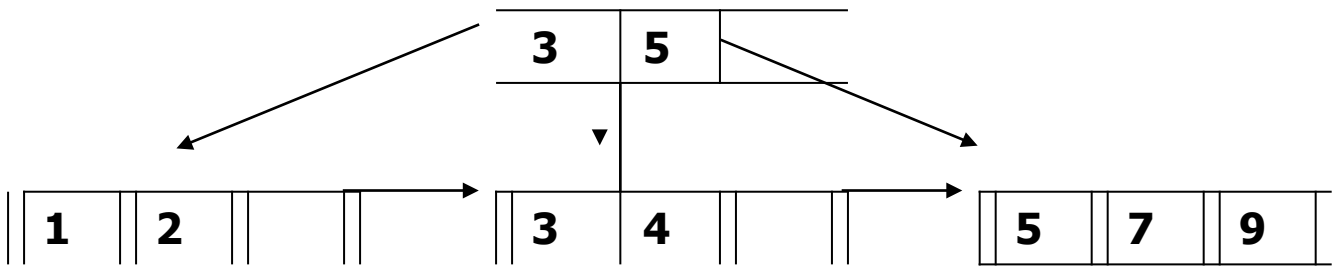
Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10



- Insert 4

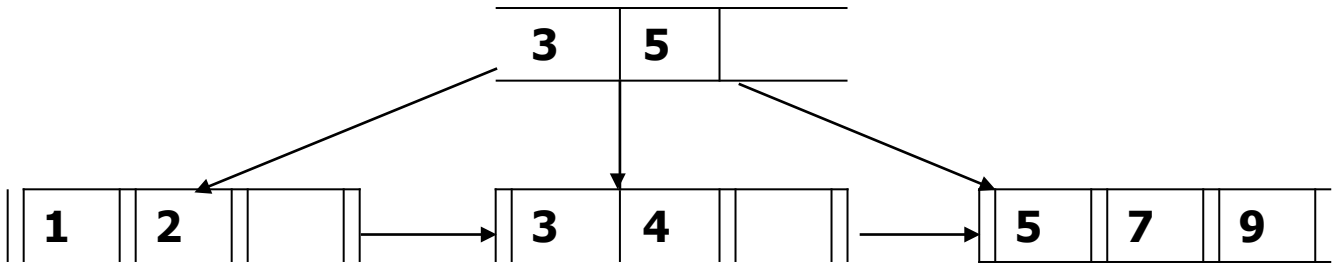


Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

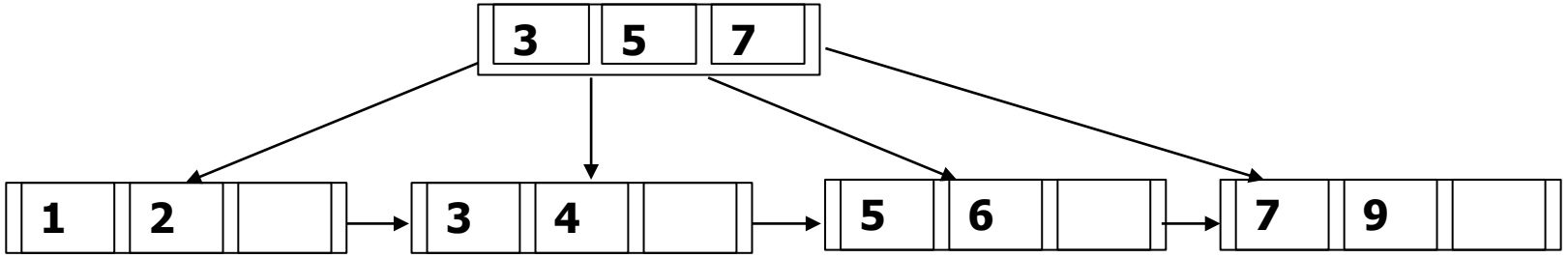


- Insert 6

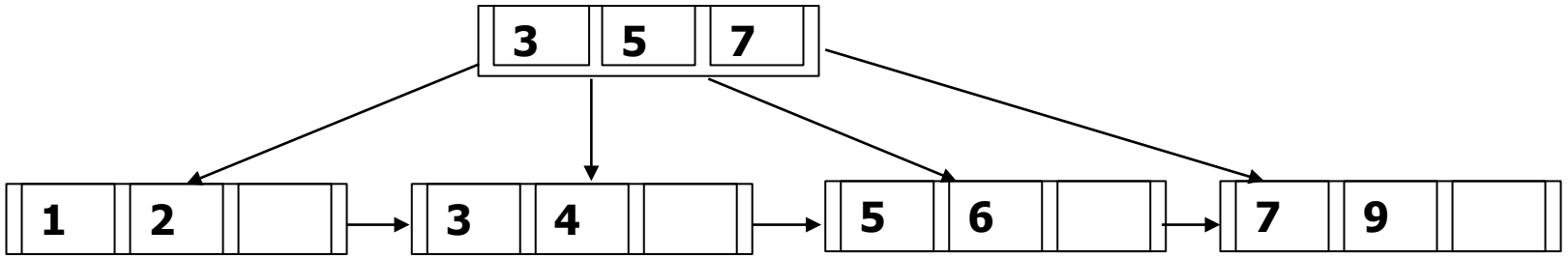
Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10



- Insert 6

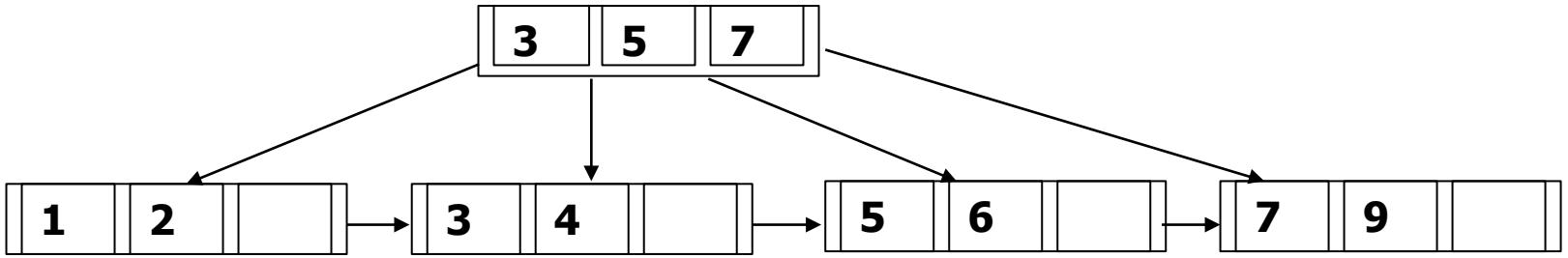


Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

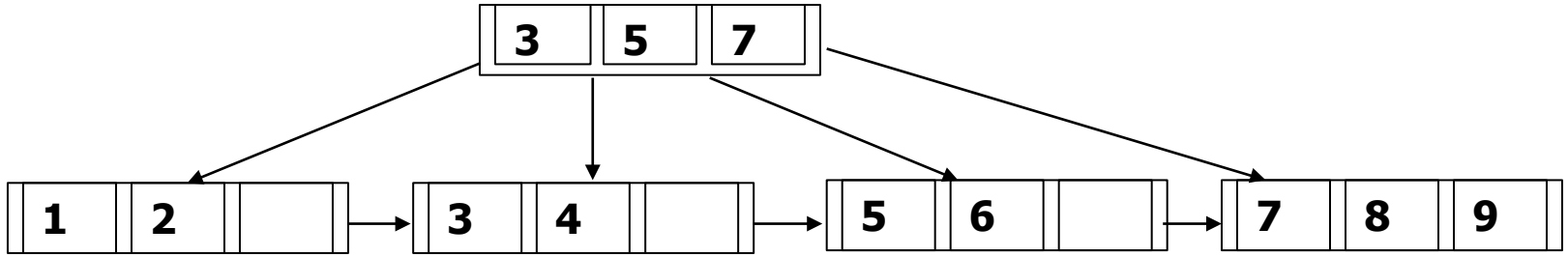


- Insert 8

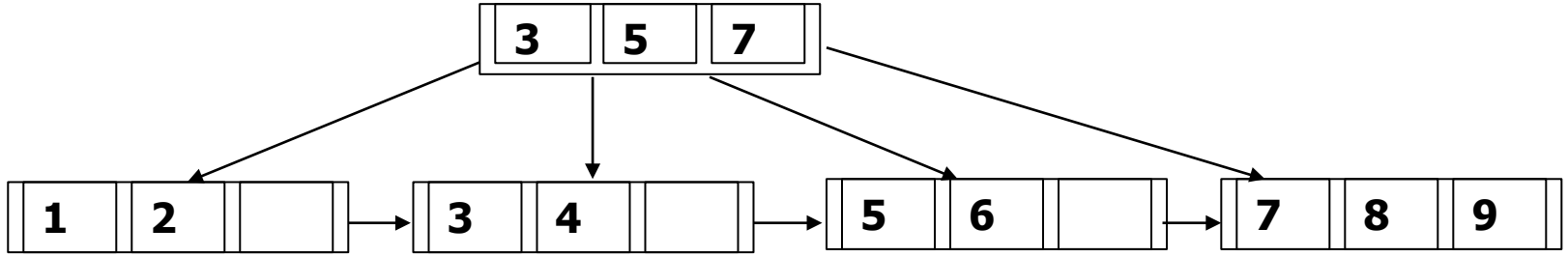
Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10



- Insert 8

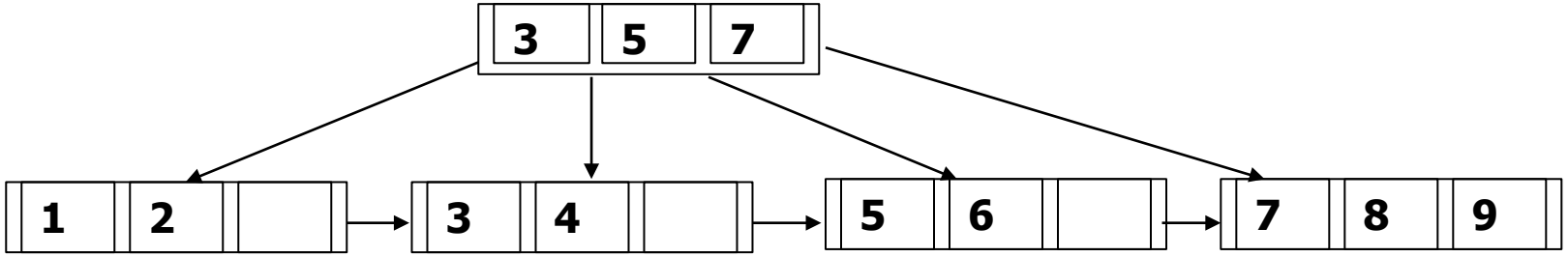


Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

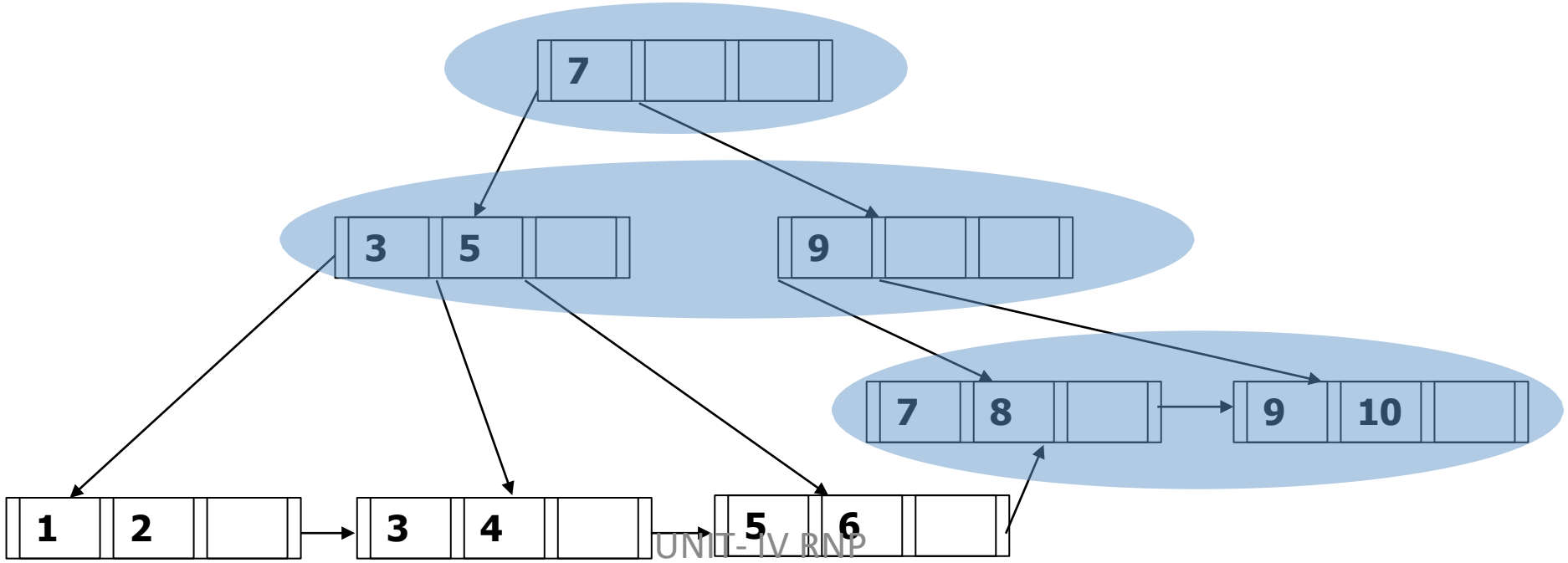


- Insert 10

Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

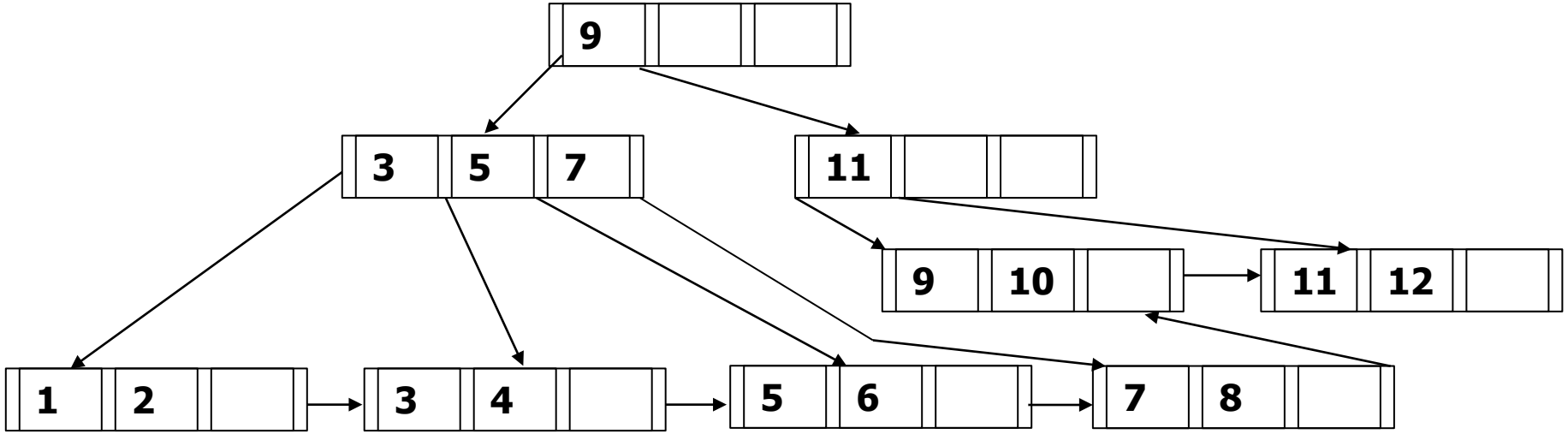


- Insert 10



- Deletion- Shown in the class

Show the tree after deletions



- Remove 9, 7, 8

Difference between B Tree and B+ Tree

B-Tree	B+ Tree
Data is stored in leaf nodes as well as internal nodes.	Data is stored only in leaf nodes.
Searching is a bit slower as data is stored in internal as well as leaf nodes.	Searching is faster as the data is stored only in the leaf nodes.
No redundant search keys are present.	Redundant search keys may be present.
Deletion operation is complex.	Deletion operation is easy as data can be directly deleted from the leaf nodes.
Leaf nodes cannot be linked together.	Leaf nodes are linked together to form a linked list.

SPLAY TREES

Dr Ratna Patil

Splay Trees

- Splay trees are binary search trees (BSTs) that:
  - Are not perfectly balanced all the time
  - Allow search and insertion operations to try to balance the tree so that future operations may run faster

- Based on the heuristic:
  - If X is accessed once, it is likely to be accessed again.
  - After node X is accessed, perform "splaying" operations to bring X up to the root of the tree.
  - Do this in a way that leaves the tree more or less balanced as a whole.

Splay Tree

- Splaying : move node to root using rotates
- Splay Tree : after any access, move node to root
  - O(1) to access same element again
  - Faster to access nearby values

Splay tree

- Splay tree is another variant of binary search tree. In a splay tree, the recently accessed element is placed at the root of the tree. A splay tree is defined as follows...

- Splay Tree is a self - adjusted Binary Search Tree in which every operation on an element rearrange the tree so that the element is placed at the root position of the tree.

- In a splay tree, every operation is performed at root of the tree. All the operations on a splay tree are involved with a common operation called **"Splaying"**.

Dr Ratna Patil

Cont..

- Splaying an element is the process of bringing it to the root position by performing suitable rotation operations.

- In a splay tree, splaying an element rearrange all the elements in the tree so that splayed element is placed at root of the tree.

- In a splay tree, to splay any element we use the following rotation operations...

Dr Ratna Patil

Rotations in Splay Tree

1. **Zig Rotation**
2. **Zag Rotation**
3. **Zig - Zig Rotation**
4. **Zag - Zag Rotation**
5. **Zig - Zag Rotation**
6. **Zag - Zig Rotation**

Dr Ratna Patil

Zig Rotation

The **Zig Rotation** in a splay tree is similar to the single right rotation in AVL Tree rotations. In zig rotation every node moves one position to thr righ from its current position. Consider the following example...

Zag Rotation

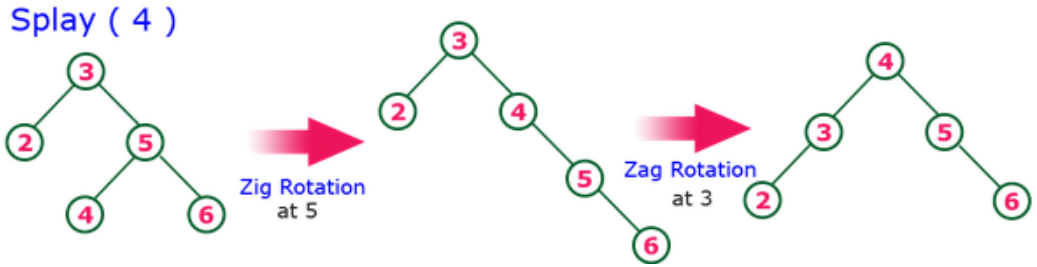
The **Zag Rotation** in a splay tree is similar to the single left rotation in AVL Tree rotations. In zag rotation every node moves one position to the left from its current position. Consider the following example...

Zag-Zag Rotation

The **Zag-Zag Rotation** in a splay tree is a double zag rotation. In zag-zag rotation every node moves two position to the left from its current position. Consider the following example...

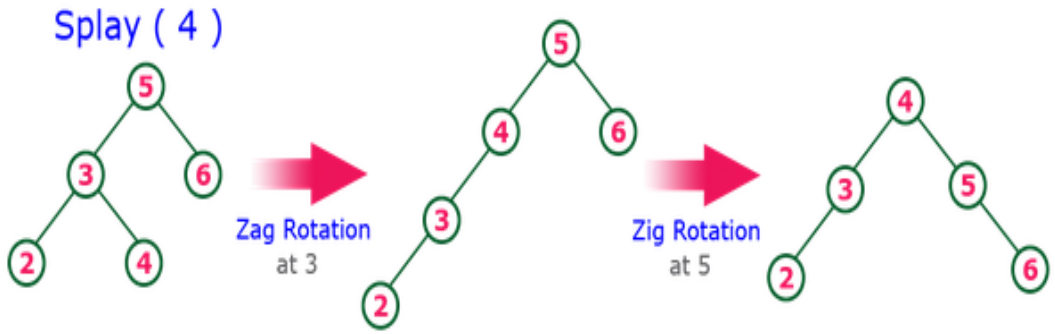
Zig-Zag Rotation

The **Zig-Zag Rotation** in a splay tree is a sequence of zig rotation followed by zag rotation. In zig-zag rotation every node moves one position to the right followed by one position to the left from its current position. Consider the following example...



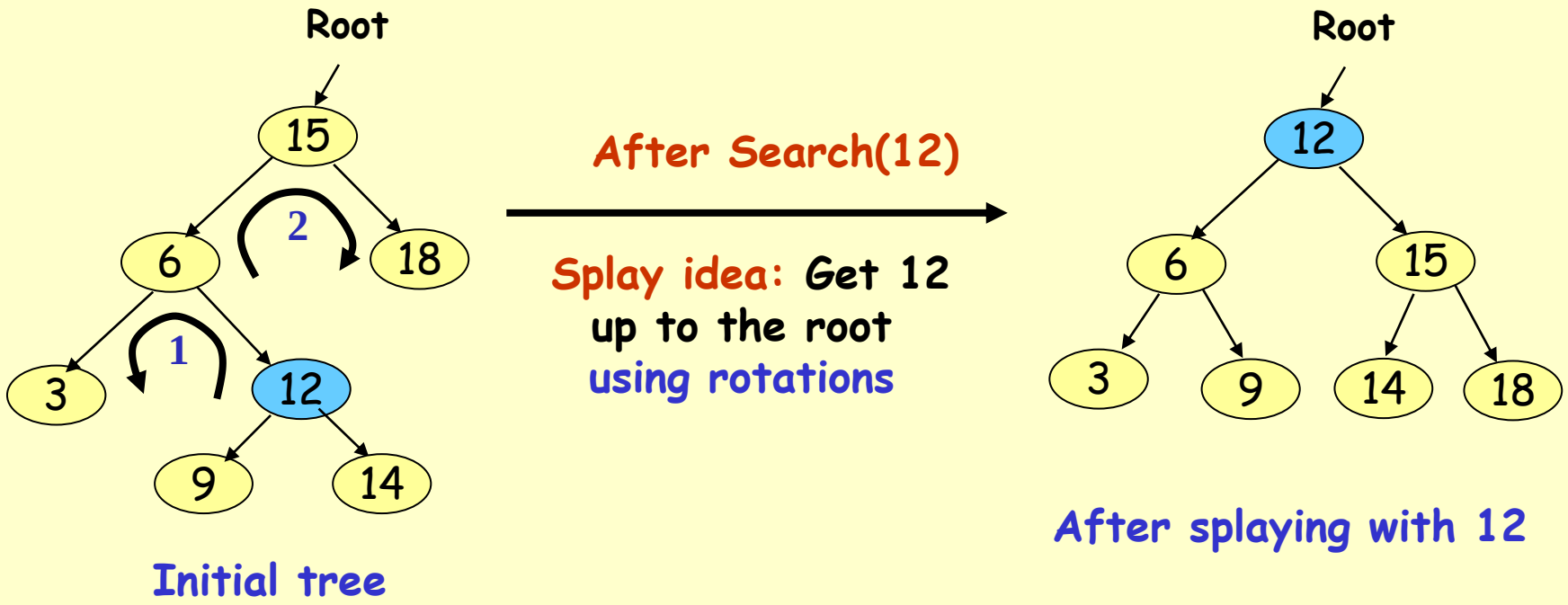
Zag-Zig Rotation

The **Zag-Zig Rotation** in a splay tree is a sequence of zag rotation followed by zig rotation. In zag-zig rotation every node moves one position to the left followed by one position to the right from its current position. Consider the following example...



Every Splay tree must be a binary search tree but it is need not to be balanced tree.

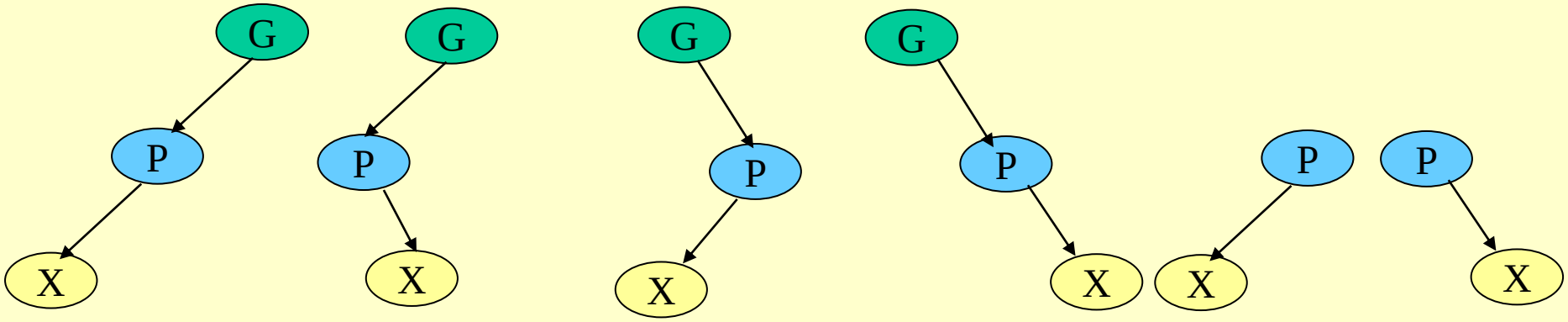
Motivating Example



- Not only splaying with 12 makes the tree balanced, subsequent accesses for 12 will take O(1) time.
- Active (recently accessed) nodes will move towards the root and inactive nodes will slowly move further from the root

Splay Tree Terminology

- Let X be a non-root node, i.e., has at least 1 ancestor.
- Let P be its parent node.
- Let G be its grandparent node (if it exists)
- Consider a path from G to X:
  - Each time we go left, we say that we "zig"
  - Each time we go right, we say that we "zag"
- There are 6 possible cases:



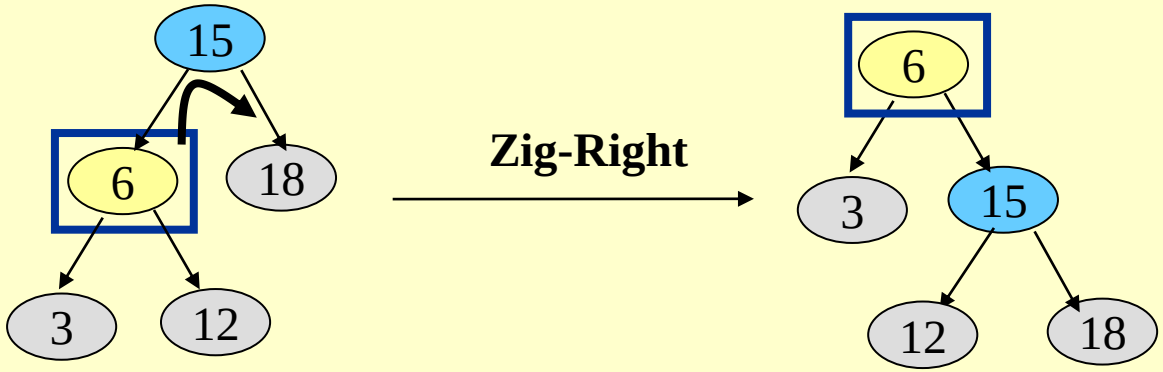
1. zig-zig 2. zig-zag 3. zag-zig 4. zag-zag 5. zig 6. zag

Splay Tree Operations

- When node X is accessed, apply one of six rotation operations:
  - Single Rotations (X has a P but no G)
    - zig, zag
  - Double Rotations (X has both a P and a G)
    - zig-zig, zig-zag
    - zag-zig, zag-zag

Splay Trees: Zig Operation

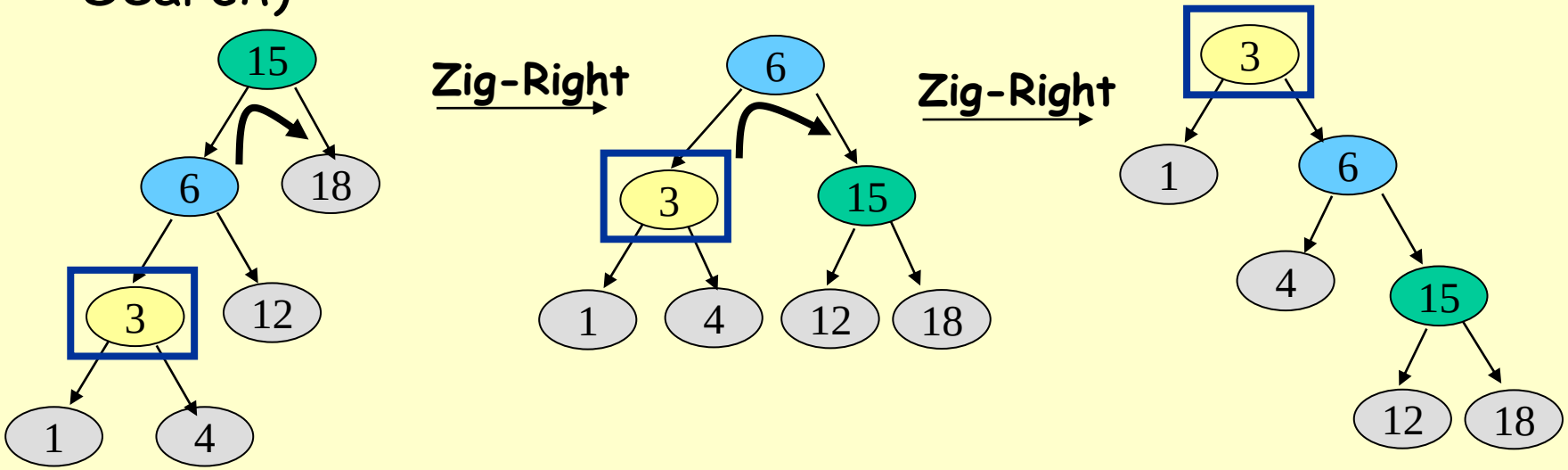
- "Zig" is just a single rotation, as in an AVL tree
- Suppose 6 was the node that was accessed (e.g. using Search)



- "Zig-Right" moves 6 to the root.
- Can access 6 faster next time: O(1)
- Notice that this is simply a right rotation in AVL tree terminology.

Splay Trees: Zig-Zig Operation

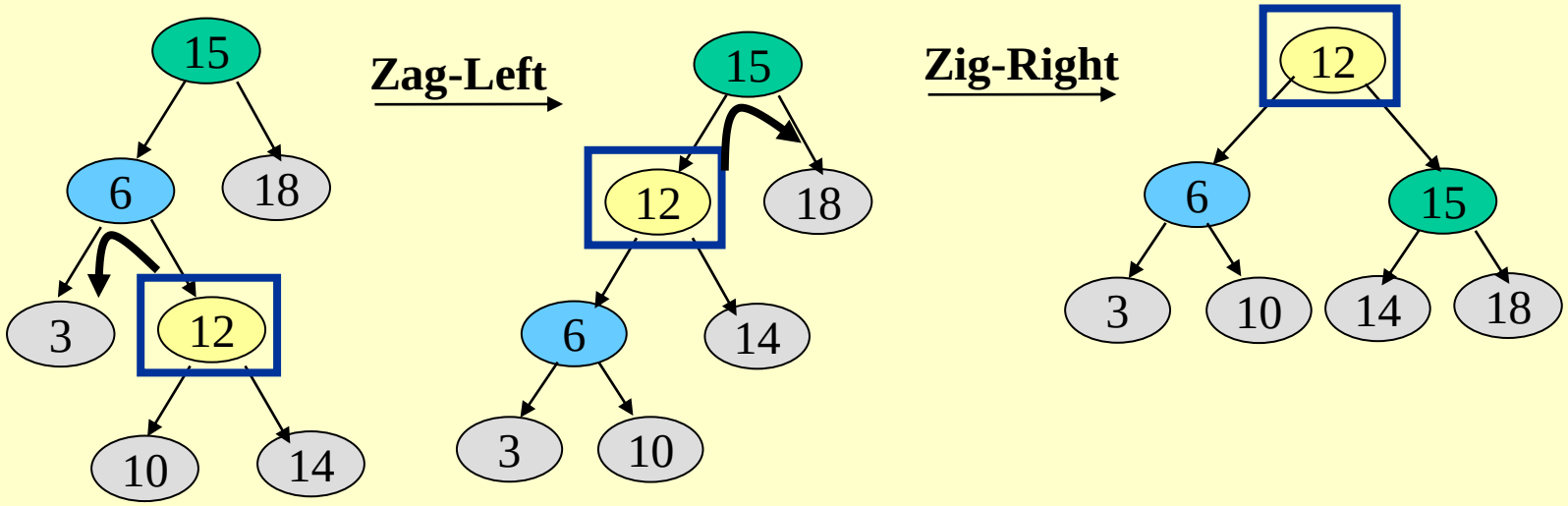
- "Zig-Zig" consists of two single rotations of the same type
- Suppose 3 was the node that was accessed (e.g., using Search)



- Due to "zig-zig" splaying, 3 has bubbled to the top!
- Note: Parent-Grandparent is rotated first.

Splay Trees: Zig-Zag Operation

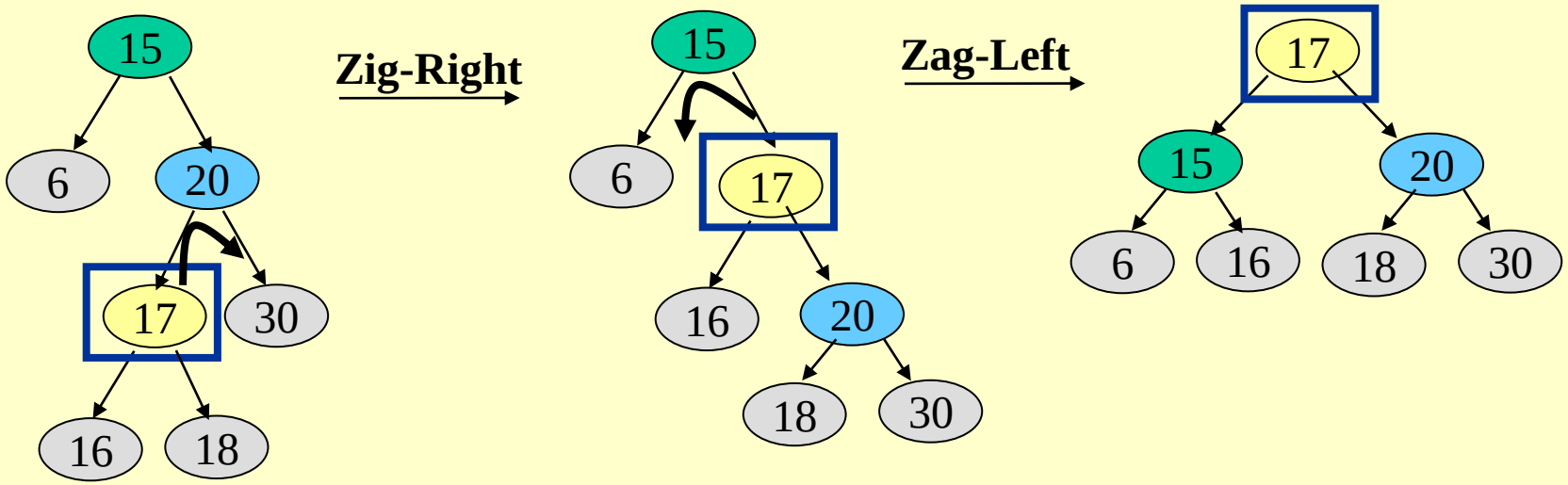
- "Zig-Zag" consists of two rotations of the opposite type
- Suppose 12 was the node that was accessed (e.g., using Search)



- Due to "zig-zag" splaying, 12 has bubbled to the top!
- Notice that this is simply an LR imbalance correction in AVL tree terminology (first a left rotation, then a right rotation)

Splay Trees: Zag-Zig Operation

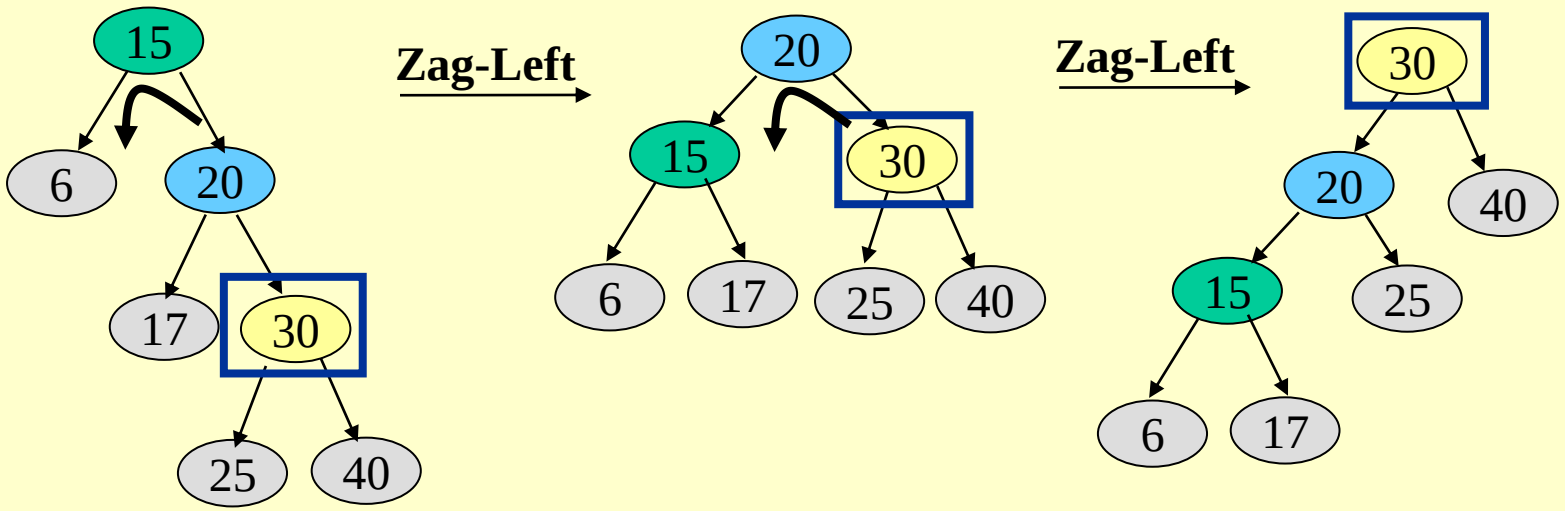
- "Zag-Zig" consists of two rotations of the opposite type
- Suppose 17 was the node that was accessed (e.g., using Search)



- Due to "zag-zig" splaying, 17 has bubbled to the top!
- Notice that this is simply an RL imbalance correction in AVL tree terminology (first a right rotation, then a left rotation)

Splay Trees: Zag-Zag Operation

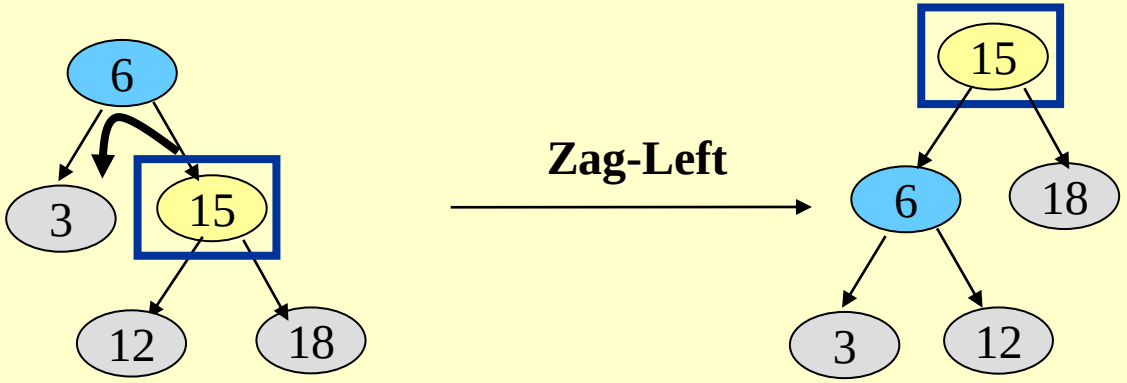
- "Zag-Zag" consists of two single rotations of the same type
- Suppose 30 was the node that was accessed (e.g., using Search)



- Due to "zag-zag" splaying, 30 has bubbled to the top!
- Note: Parent-Grandparent is rotated first.

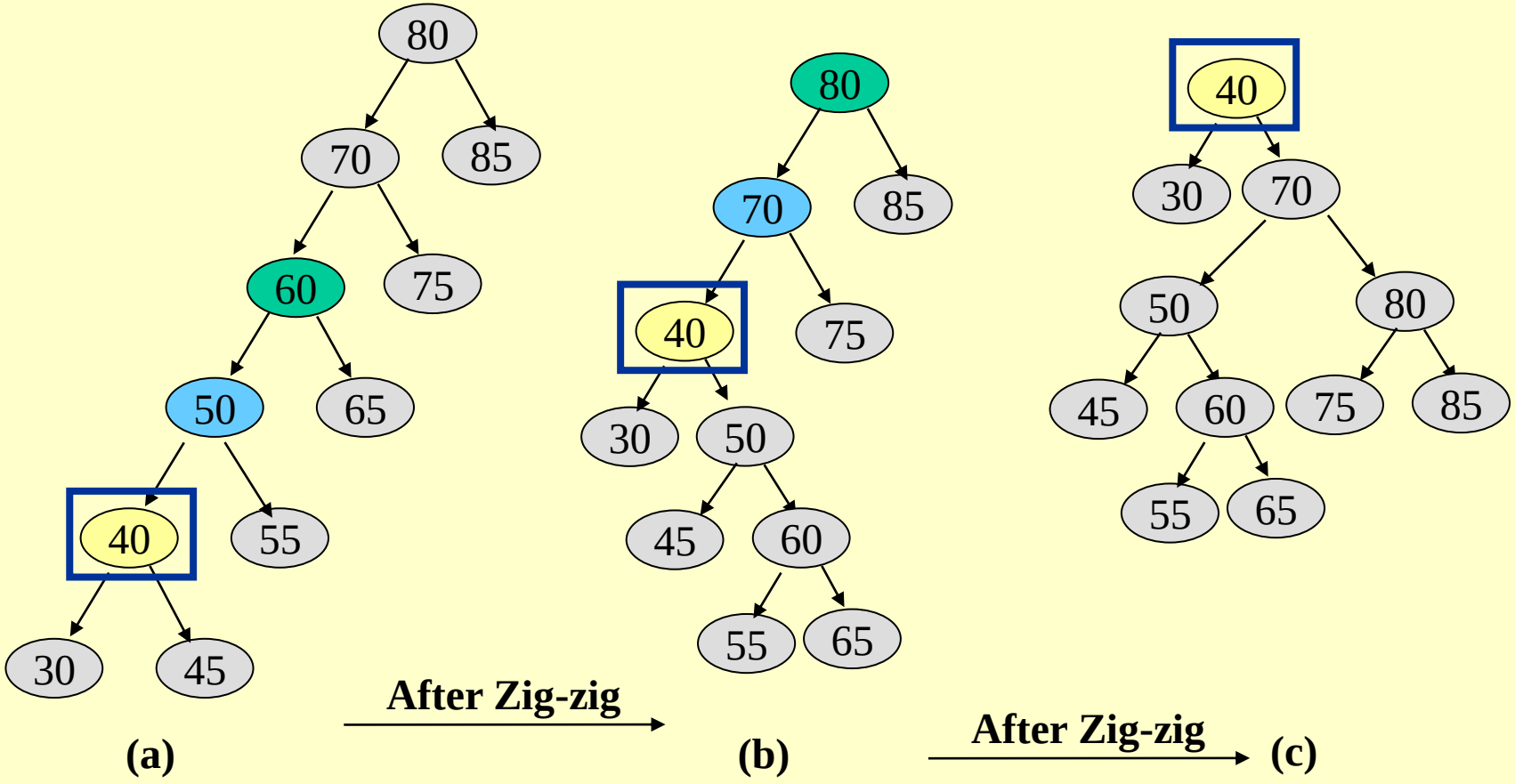
Splay Trees: Zag Operation

- "Zag" is just a single rotation, as in an AVL tree
- Suppose 15 was the node that was accessed (e.g., using Search)

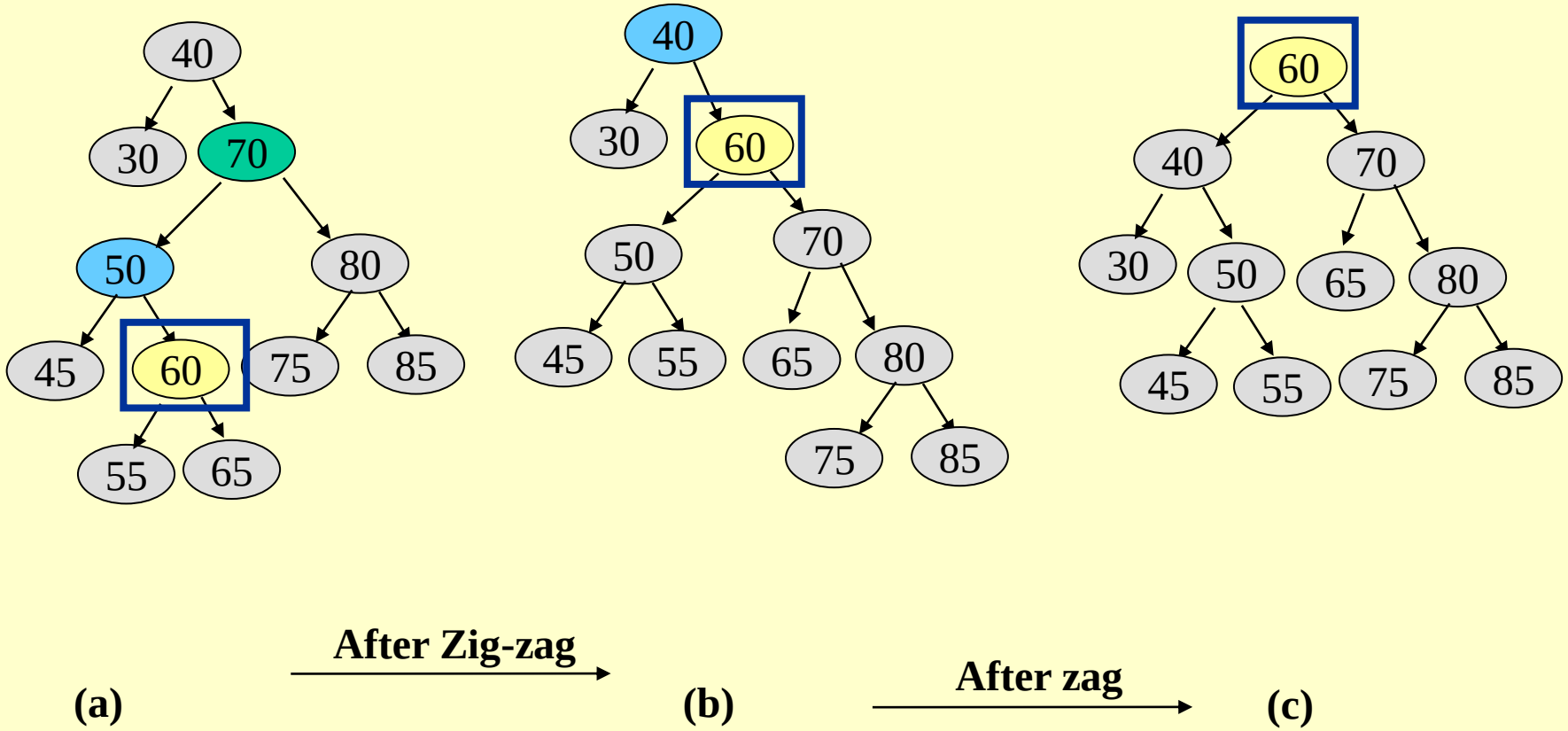


- "Zag-Left"moves 15 to the root.
- Can access 15 faster next time: O(1)
- Notice that this is simply a left rotation in AVL tree terminology

Splay Trees: Example – 40 is accessed



Splay Trees: Example – 60 is accessed



Insertion

- To insert a value *x* into a splay tree:
  - Insert *x* as with a normal binary search tree.
  - when an item is inserted, a splay is performed.
  - As a result, the newly inserted node *x* becomes the root of the tree.

Dr Ratna Patil

Insertion Operation in Splay Tree

- The insertion operation in Splay tree is performed using following steps...
- **Step 1:** Check whether tree is Empty.
- **Step 2:** If tree is Empty then insert the **newNode** as Root node and exit from the operation.
- **step 3:** If tree is not Empty then insert the newNode as a leaf node using Binary Search tree insertion logic.
- **Step 4:** After insertion, **Splay** the **newNode**

Dr Ratna Patil

Let’s solve it!

- Insert

5, 10, 15, 6, 9,7,11,4,12

Dr Ratna Patil

Deletion

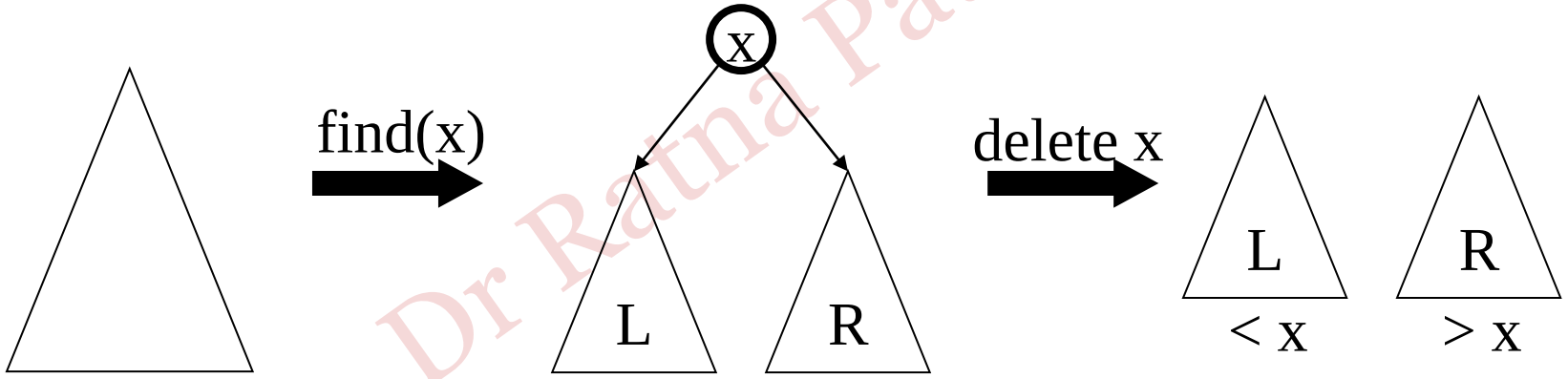
- Delete x
  - Splay x to root and remove it. (note: the node does not have to be a leaf or single child node like in BST delete.) Two trees remain, right subtree and left subtree.
  - Splay the max in the left subtree to the root
  - Attach the right subtree to the new root of the left subtree.

Dr Rathna Patil

Delete(x, T):

∗ Splay(x, T) and remove root → tree falls into T1 and T2.
∗ Splay(x, T1)
∗ Make T2 right son of new root of T1 after splay

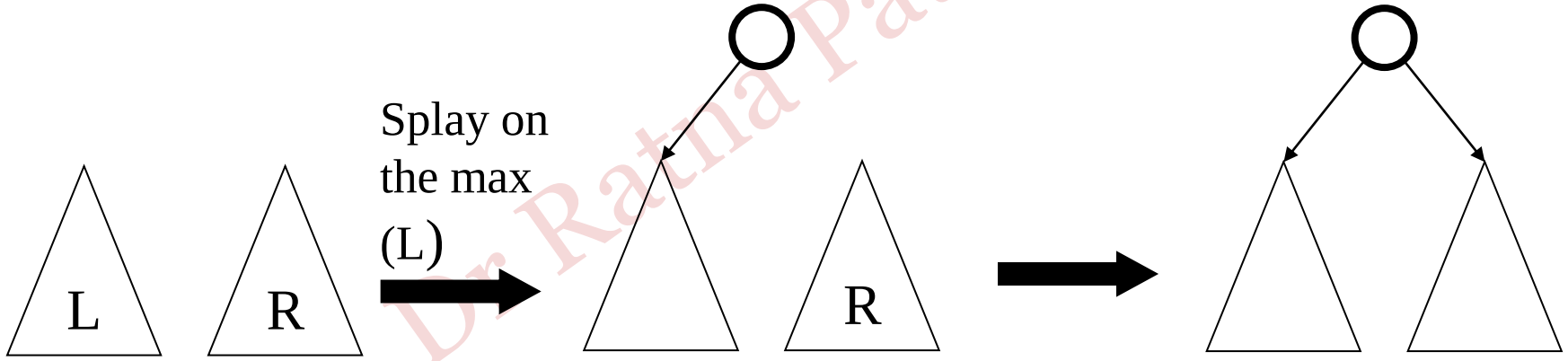
Deletion (cont.)



Dr Ratna Patil

Join

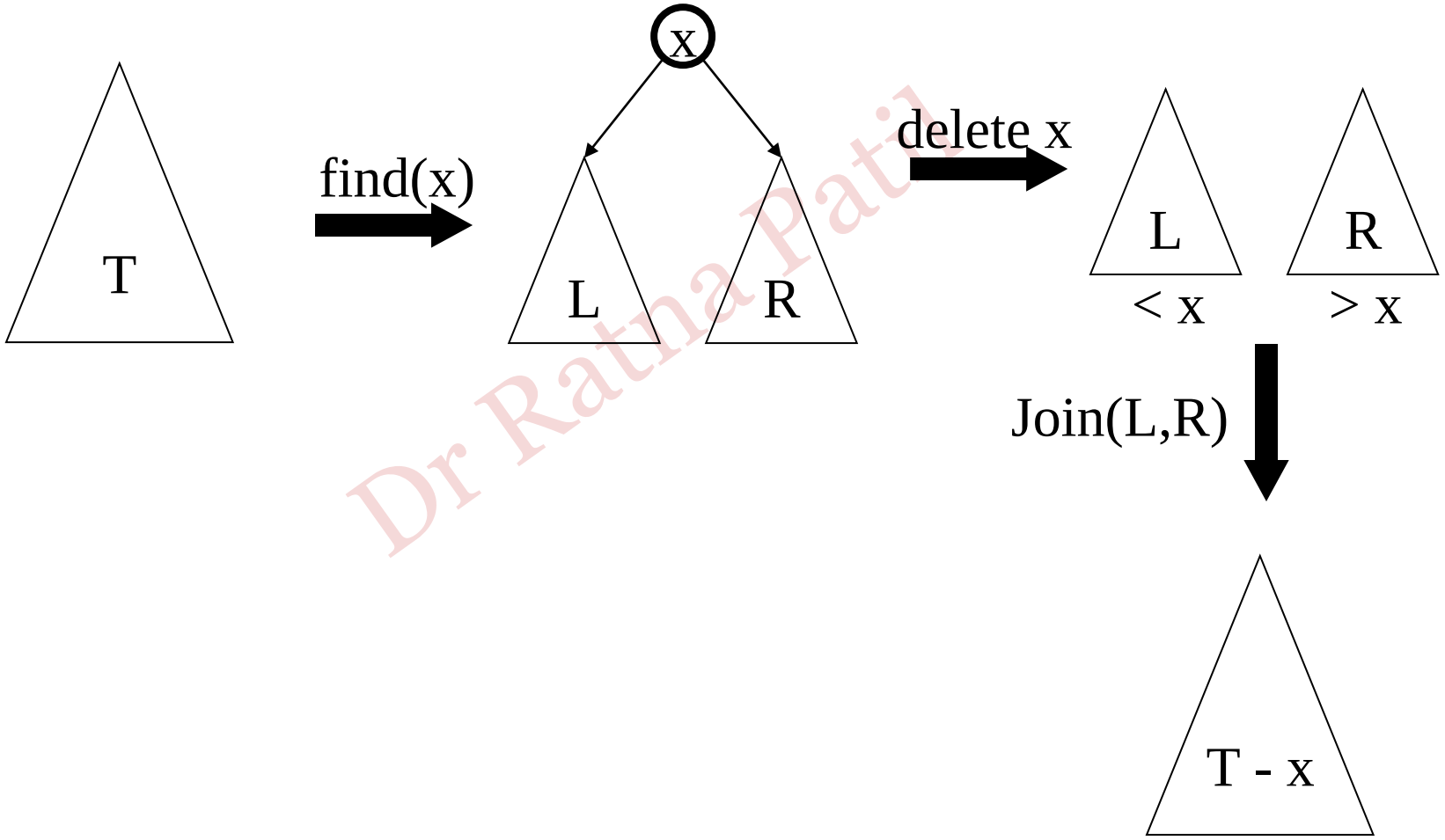
Join(L, R): given two trees such that L < R, merge them



Dr Ratna Patil

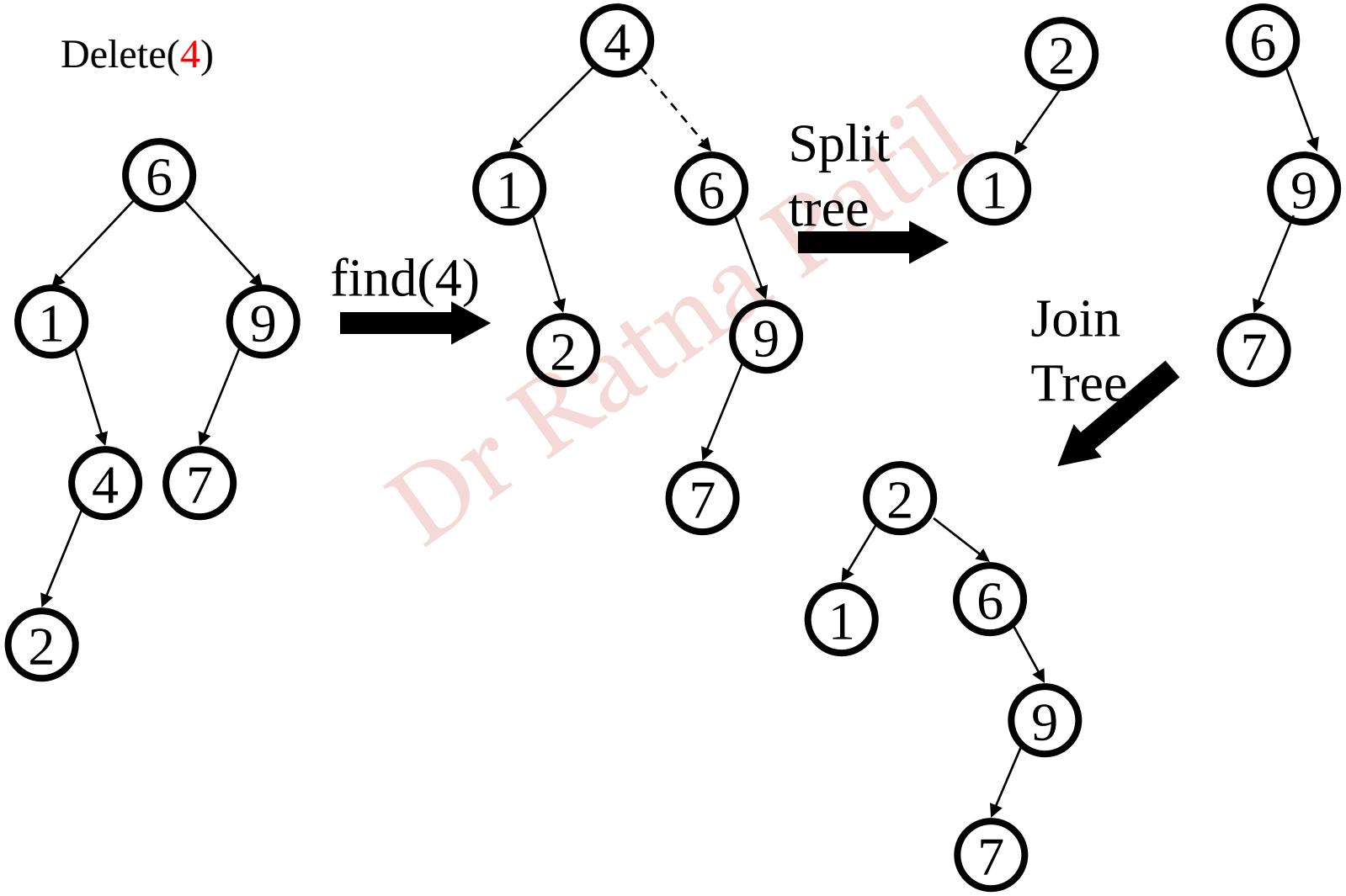
Splay on the maximum element in L, then attach R

Deletion (cont.)



Dr Ratna Patil

DELETE EXAMPLE



Dr Ratna Patil

Red Black Binary Tree

Dr Rathna Patil

Why & What is mean by Red-Black Tree?

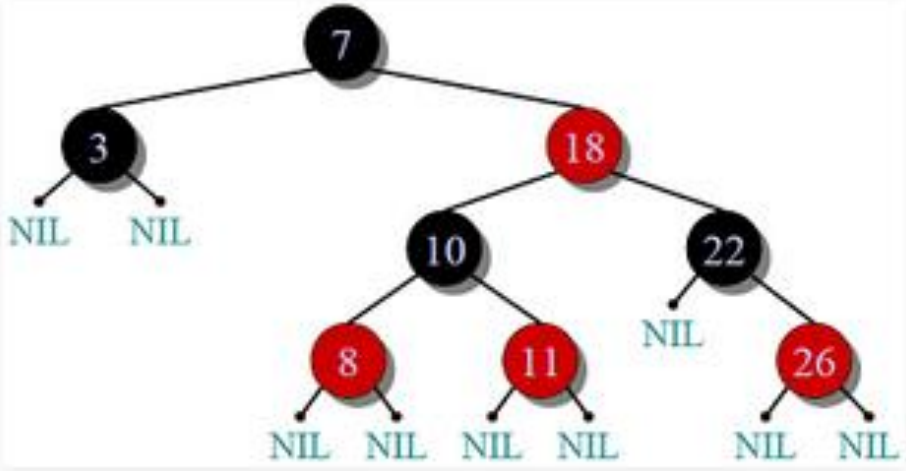
- Red-Black Tree is a self-balancing Binary Search Tree (BST).
- Every node is colored with Red or Black.

Dr Ratna Patil

Why Red Black Tree?

- Most of the BST operations (e.g., search, max, min, insert, delete.. etc) take O(h) time where h is the height of the BST. The cost of these operations is high.
- The height of a Red-Black tree is always O(Log n) where n is the number of nodes in the tree.

Properties of Red Black Tree



1. Every node has a color either Red or Black.
2. Root of tree is always Black.
3. Every leaf (NULL pointer) is black
4. Children's of Red Node are Black, Parent of Red Node is Black. (No adjacent nodes can be Red or No Red-Red partent child relationship)
5. Every path from a node (including root) to any of its descendant NULL node has the same number of Black nodes.

Every Red Black Tree is a binary search tree but every Binary Search Tree need not be Red Black tree.

Dr Ratna Patil

Insertion in Red-Black Tree: (Conditions)

1. Tree is empty – Add root node with color Black.
2. Tree is not empty – Add new node with color Red.
3. If parent of newnode is Black then exit.
4. If parent of newnode is Red then check the color of parent's siblings of new node-

   a. If parent of sibling is Black or NULL then perform suitable Rotation & Recolor.

   b. If parent of sibling is Red then Recolor and also check if parents parent of newnode is not root node then Recolor it and Check it

Dr Ratna Patil

Rotation Means?

Rotations: L-Left & R-Right

- It occurs on only newnode and There Parent's Node.

1. **LR Relationship** – 1. Left Rotation 2. Right Rotation
2. **RL Relationship** – 1. Right Rotation 2. Left Rotation

Dr Ratna Patil

It occurs on newnode , there Parent's Node and also there Parent's node.

1. **LL Relationship** – 1. Right Rotation
2. **RR Relationship** – 1. Left Rotation

RBT Insertion Examples (Add data according BST property)

1. Tree is empty – Add root node with color Black.
2. Tree is not empty – Add new node with color Red.
3. If parent of newnode is Black then exit.
4. If parent of newnode is Red then check the color of parent's siblings of new node-

a. If sibling of parent is Black or NULL then perform suitable rotation & Recolor.

b. If sibling of parent is Red then Recolor and also check if parents parent of newnode is not root node then Recolor it and Check it

Create Red-Black Tree for

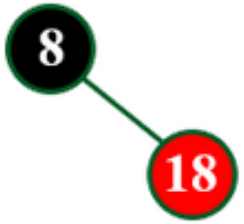
8,18,5,15,17,25,40 ,80.

insert (8)

Tree is Empty. So insert newNode as Root node with black color.

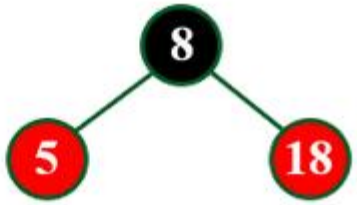
insert (18)

Tree is not Empty. So insert newNode with red color.



insert (5)

Tree is not Empty. So insert newNode with red color.



Dr Ratna P

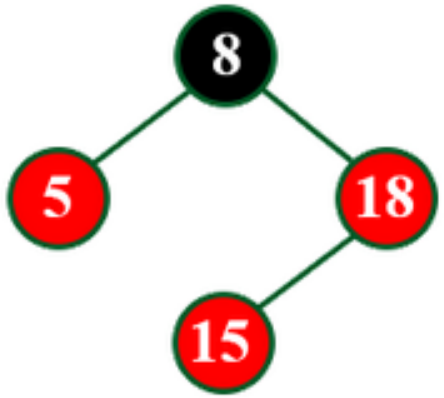
Rules

- Root node is colored **Black**.
- All the other new nodes are colored **Red**.
- If parent node is **black**, then exit.
- If parent node is **Red**, check the color of parent's sibling.
  - If uncle is **Red**, change the color of parent and uncle to **Black** and grandparent to **Red** only if it is not the root node.
  - If uncle is **Black** or Leaf node, perform rotation and recoloring.
    - **Rotation:** LL, RR, LR or RL rotation.
    - **Recoloring:** If LL or RR rotation is done, swap color of parent and grandparent. If LR or RL rotation is done, swap color of grandparent and X.

Continue..

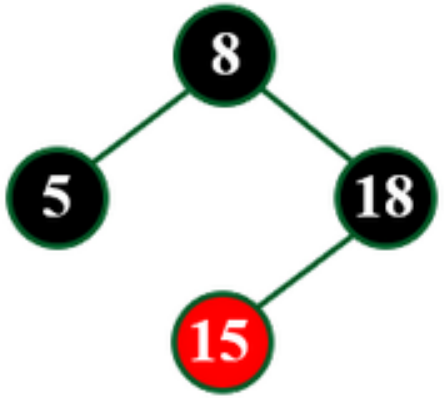
insert (15)

Tree is not Empty. So insert newNode with red color.



Here there are two consecutive Red nodes (18 & 15).
The newnode's parent sibling color is Red
and parent's parent is root node.
So we use RECOLOR to make it Red Black Tree.

After RECOLOR

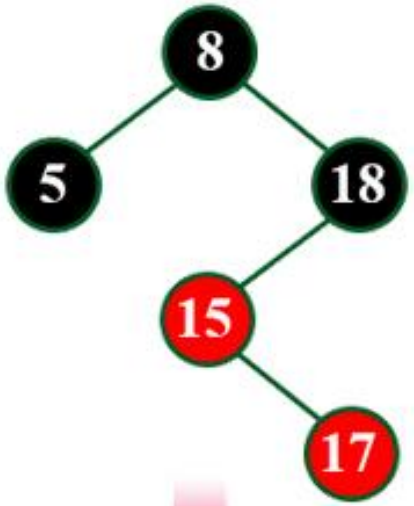


After Recolor operation, the tree is satisfying all Red Black Tree properties.

Continue..

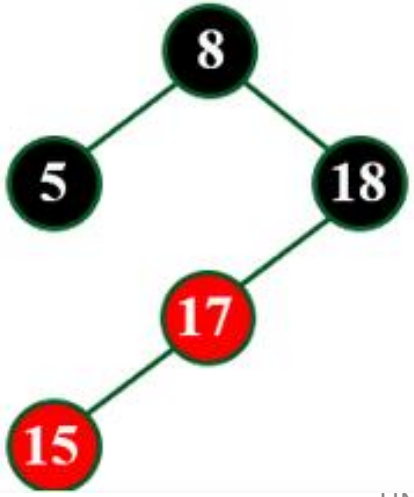
insert (17)

Tree is not Empty. So insert newNode with red color.

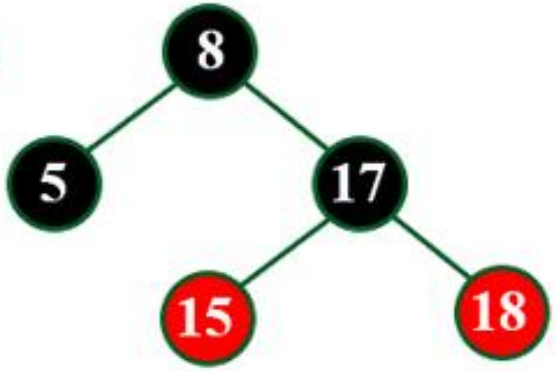


Here there are two consecutive Red nodes (15 & 17). The newnode's parent sibling is NULL. So we need rotation.
Here, we need LR Rotation & Recolor.

After Left Rotation



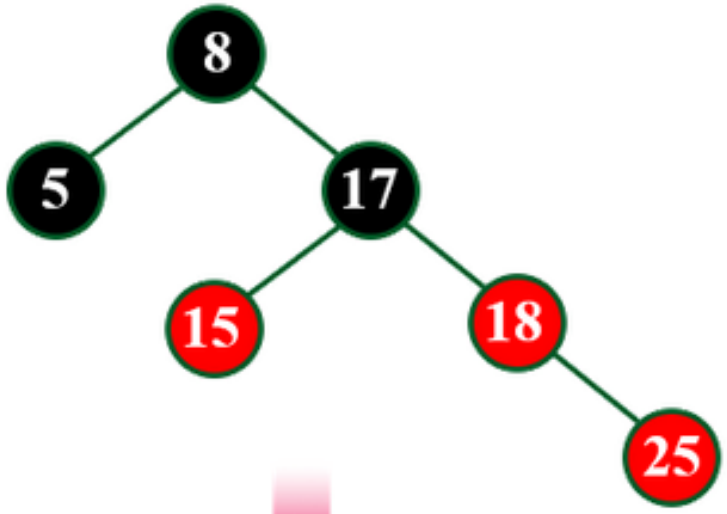
After Right Rotation & Recolor



Continue..!

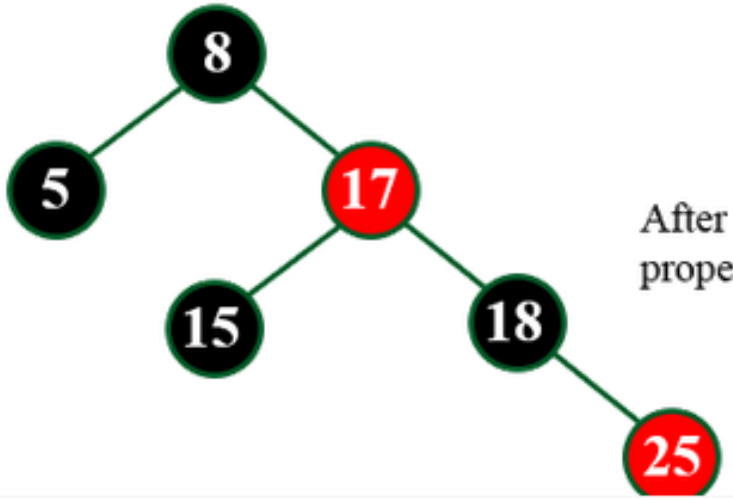
insert (25)

Tree is not Empty. So insert newNode with red color.



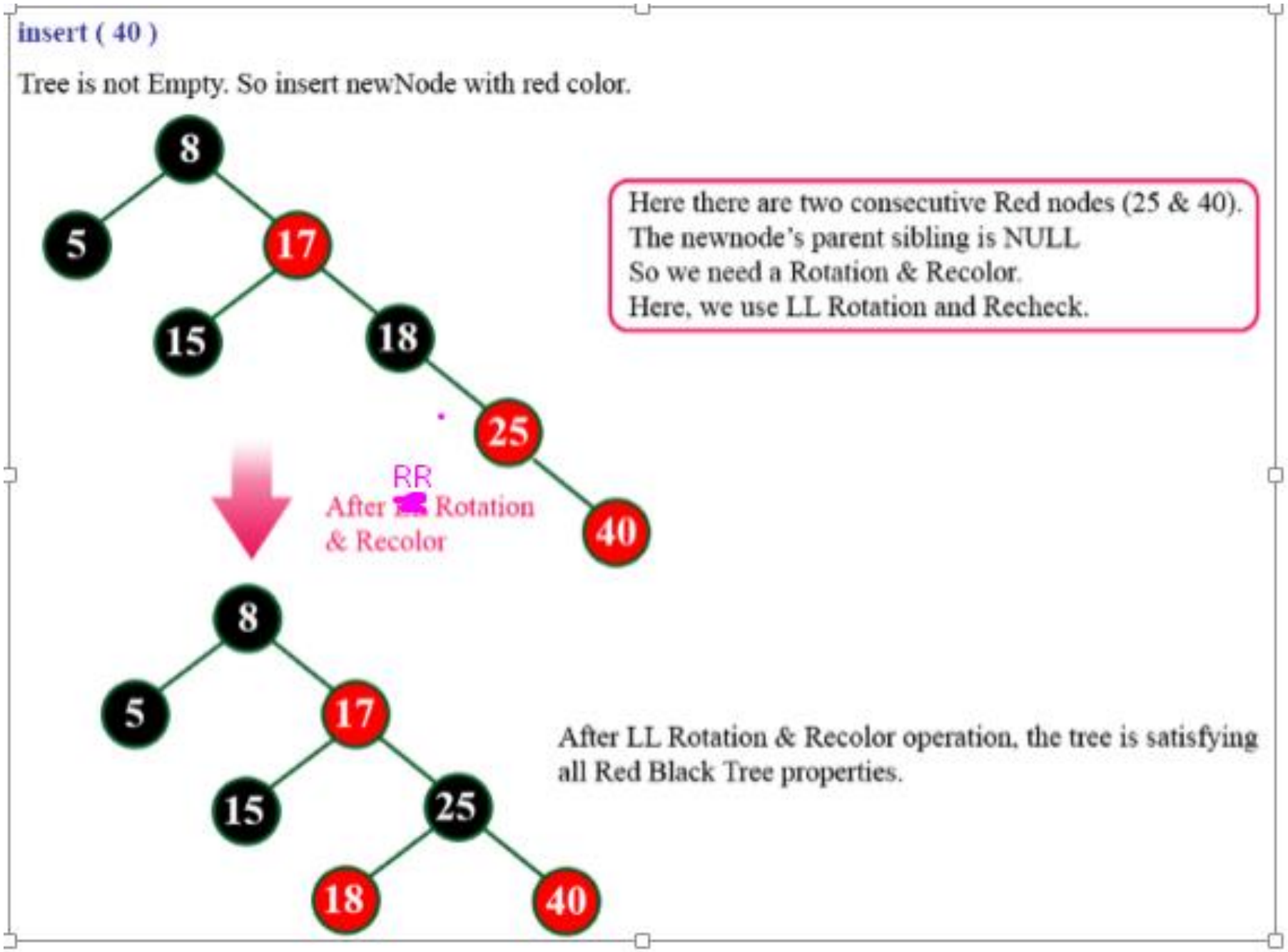
Here there are two consecutive Red nodes (18 & 25).
The newnode's parent sibling color is Red
and parent's parent is not root node.
So we use RECOLOR and Recheck.

After Recolor



After Recolor operation, the tree is satisfying all Red Black Tree properties.

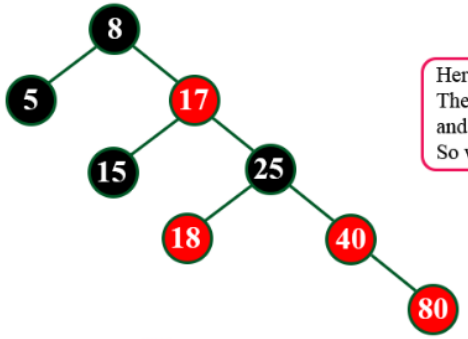
Continue..!


insert (40)
Tree is not Empty. So insert newNode with red color.
8
5
17
15
18
25
40
Here there are two consecutive Red nodes (25 & 40).
The newnode's parent sibling is NULL
So we need a Rotation & Recolor.
Here, we use LL Rotation and Recheck.
RR
After Rotation
& Recolor
8
5
17
15
25
18
40
After LL Rotation & Recolor operation, the tree is satisfying
all Red Black Tree properties.

Continue..!

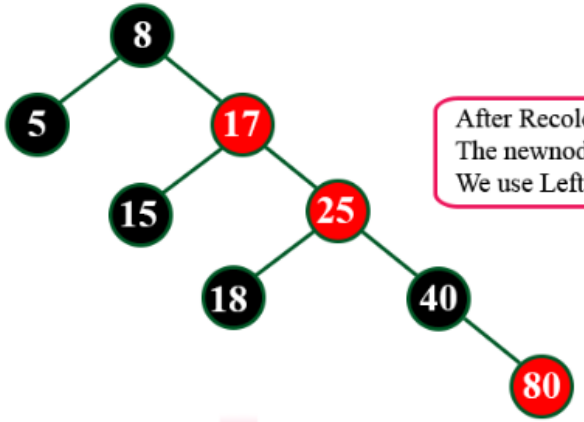
insert (80)

Tree is not Empty. So insert newNode with red color.



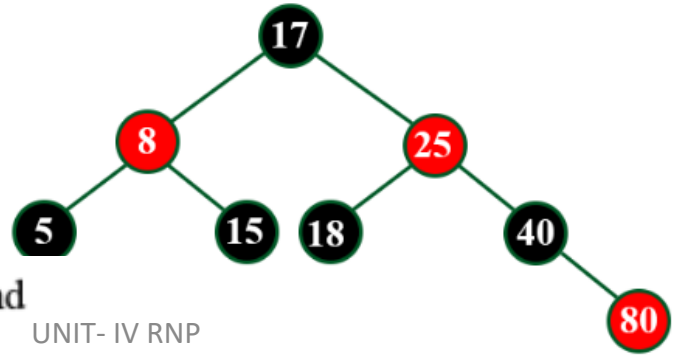
Here there are two consecutive Red nodes (40 & 80).
The newnode's parent sibling color is Red
and parent's parent is not root node.
So we use RECOLOR and Recheck.

After Recolor



After Recolor again there are two consecutive Red nodes (17 & 25).
The newnode's parent sibling color is Black. So we need Rotation.
We use Left Rotation & Recolor.

After Left Rotation
& Recolor

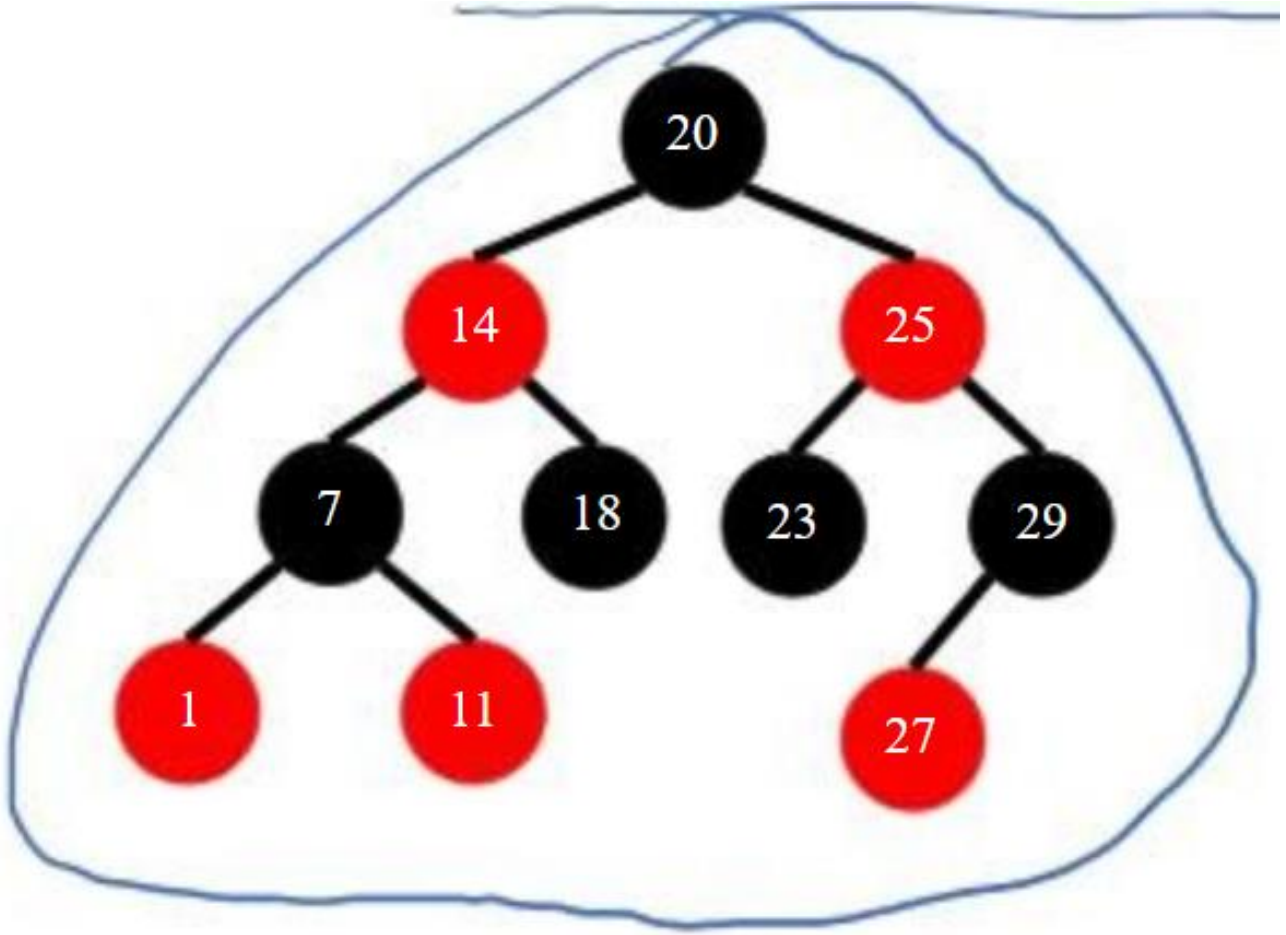


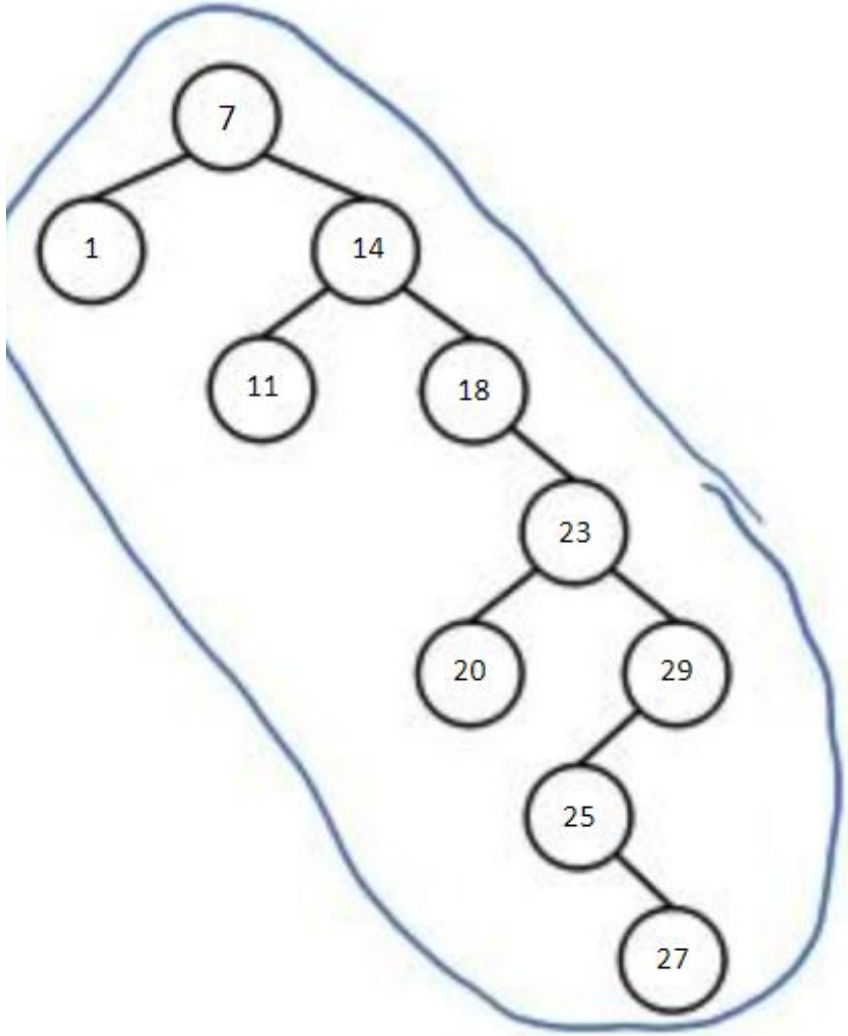
Finally above tree is satisfying all the properties of Red Black Tree and it is a perfect Red Black tree.

Compare RBT to Regular BST for Insertions: 7, 14, 18, 23, 1, 11, 20, 29, 25, 27

Dr Ratna Patil

Compare to Regular BST Insertions: 7, 14, 18, 23, 1, 11, 20, 29, 25, 27





Differences between AVL & Red-Black tree

AVL Tree	RBT Tree
Strictly height-balanced tree.	Roughly height-balanced tree.
Comparatively complex to implement.	Easy to implement.
Takes more processing for balancing.	Takes less processing for balancing because a maximum of two rotations are required.
Searching operation is faster because of height-balanced structure.	Searching operation is comparatively slow.
Insertion and deletion operations are slow.	Insertion and deletion operations are faster.
No colour for the nodes.	The nodes can be either red or black.
Balace factor is associated with each node.	There's no balance factor for nodes.
Used in databases for faster retrievals.	Used in the language libraries.

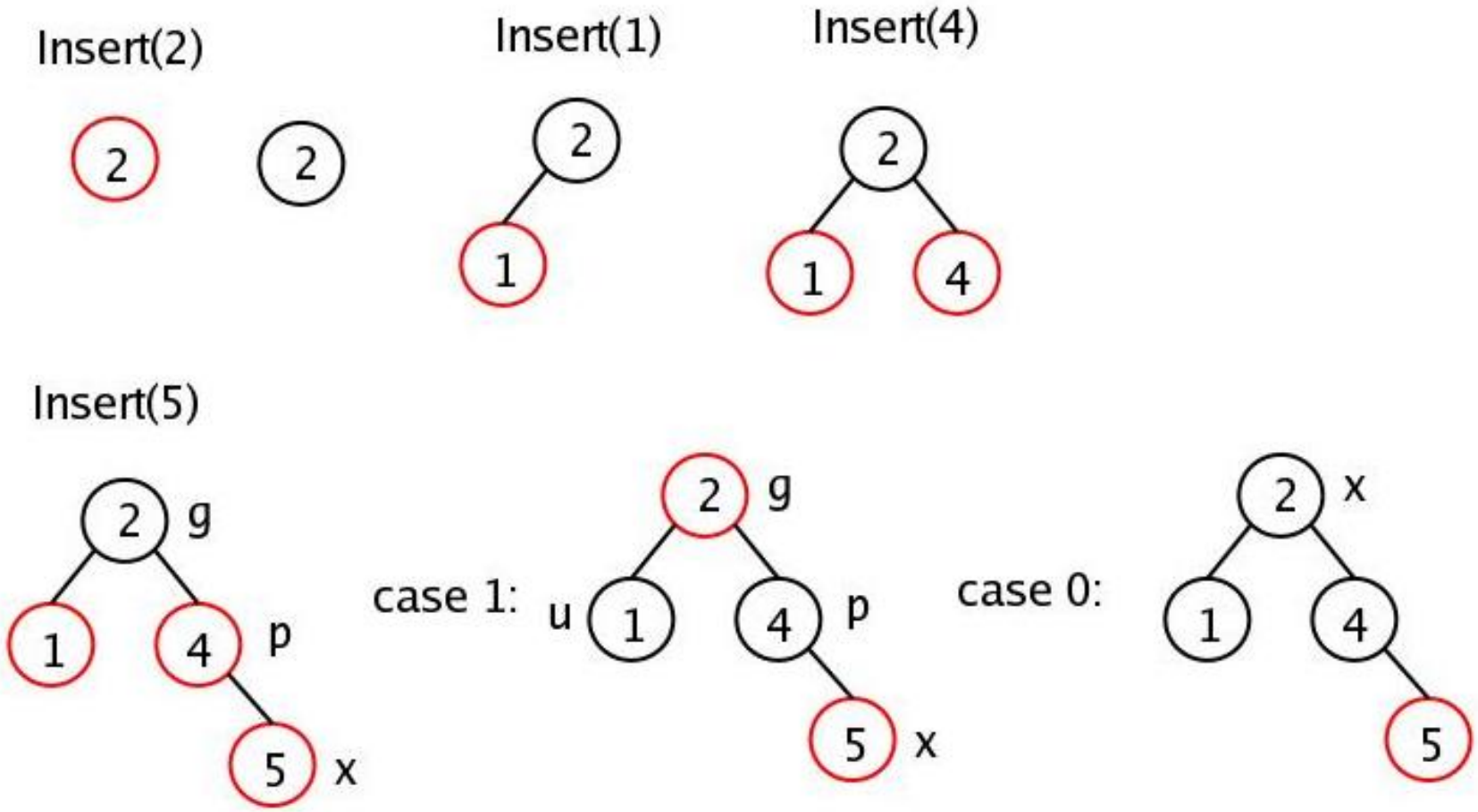
Applications of Red-Black Tree

1. Linux Kernal, For process scheduler or for keeping track of virtual memory segmentation segments for a process.
2. Used in map, multimap, multiset from C++ STL and java.util.TreeMap , java.util.TreeSet from Java.
3. They are use in K-mean clustering algorithm for reducing time complexity.
4. MySQL uses Red-Black Trees for indexes on tables.
5. Fast searching.

Dr Ratna Patil

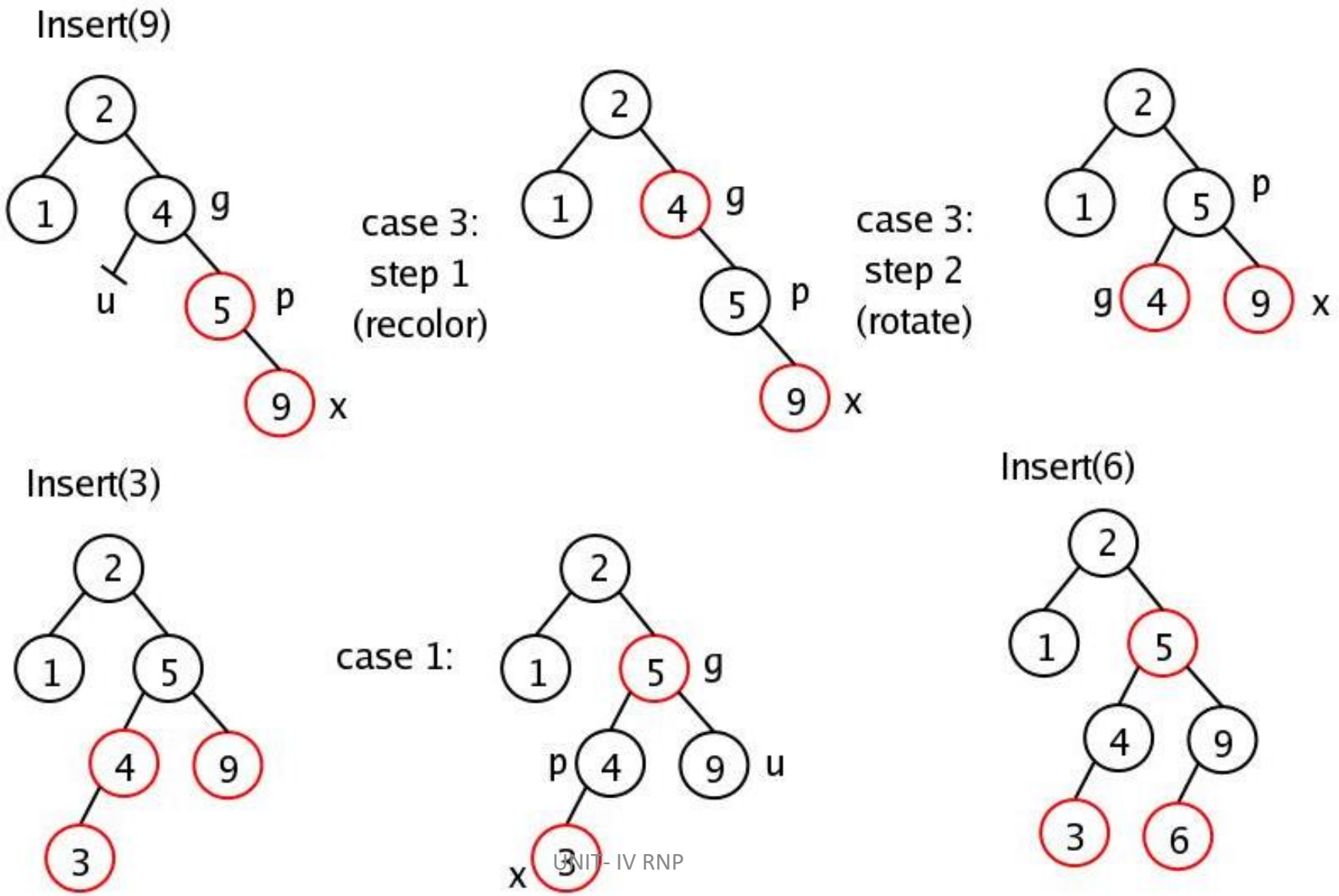
Construct Red Black tree by inserting the following elements in the given order.

Insert 2, 1, 4, 5, 9, 3, 6, 7



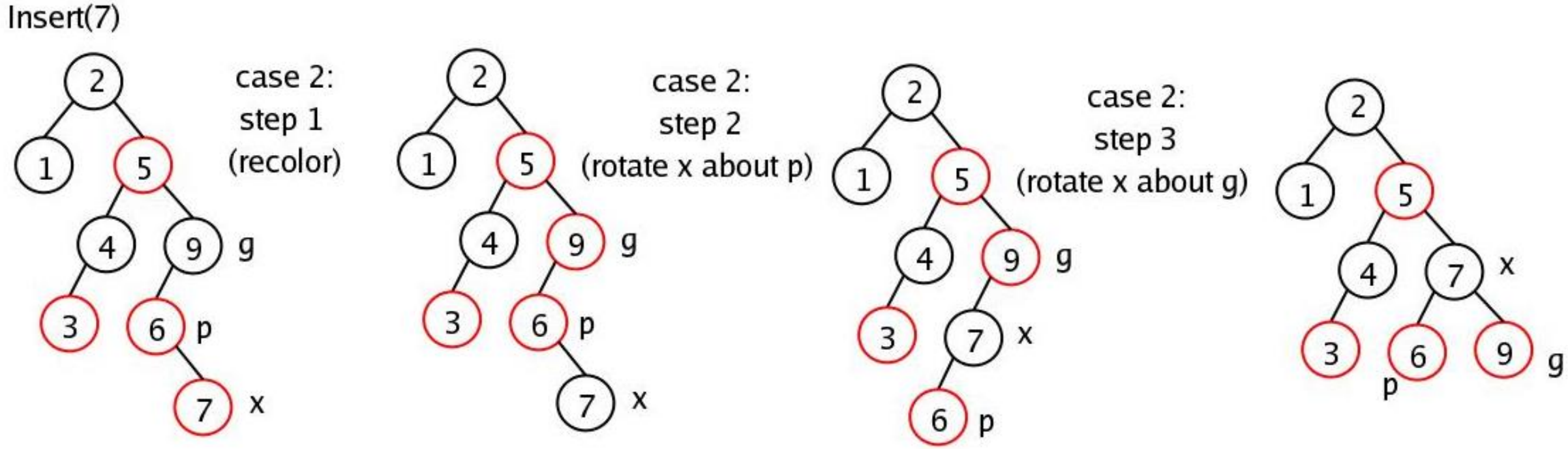
Construct Red Black tree by inserting the following elements in the given order.

Insert 2, 1, 4, 5, 9, 3, 6, 7

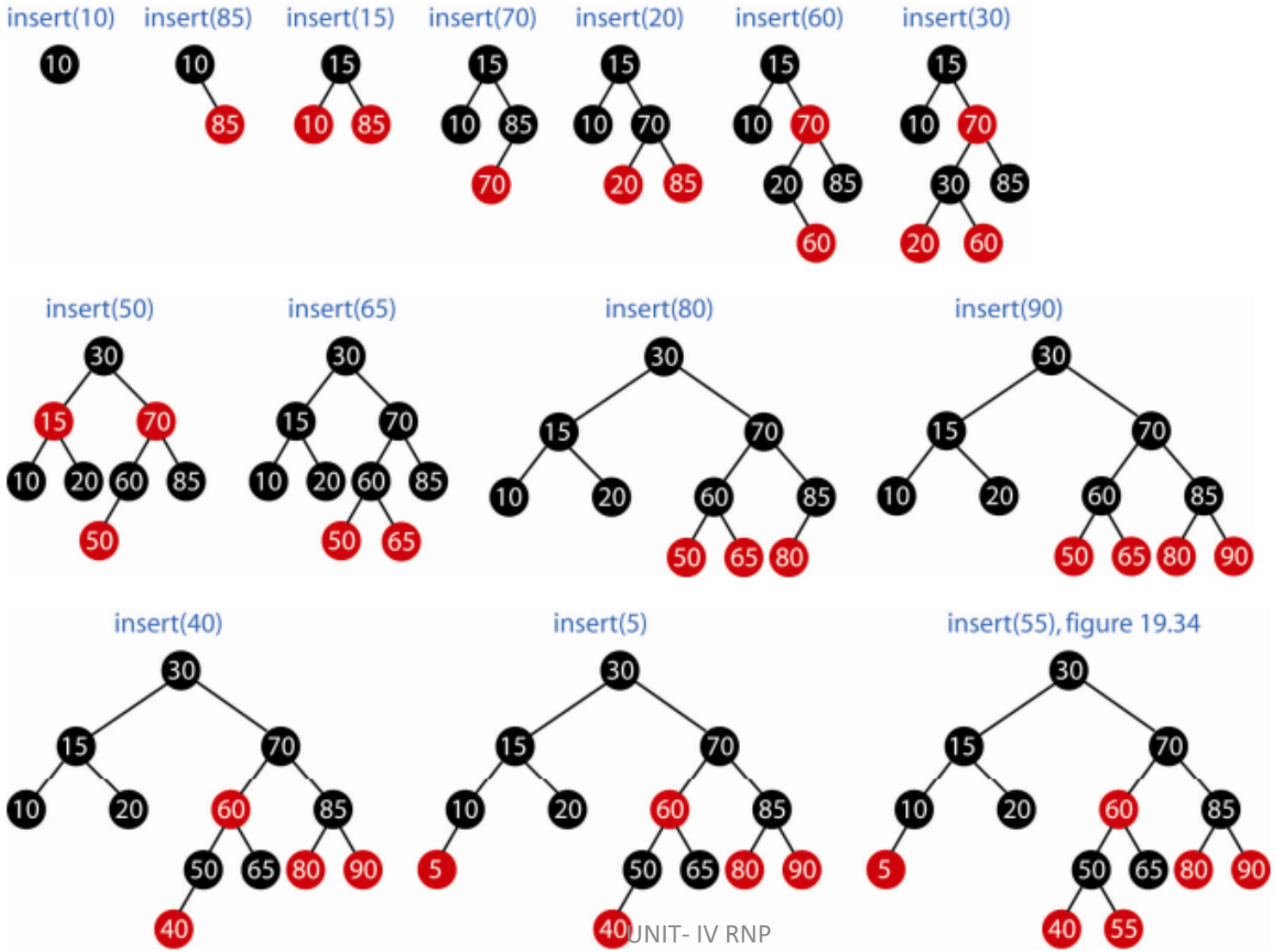


Construct Red Black tree by inserting the following elements in the given order.

Insert 2, 1, 4, 5, 9, 3, 6, 7



Example: Build a red-black tree by inserting these nodes in the following order: 10, 85, 15, 70, 20, 60, 30, 50, 65, 80, 90, 40, 5, 55. The results of each step are shown below. The final tree is figure 19.34 from the text.



Important Questions

1. Write a short note on Red-Black Tree? 6M
2. Explain Red-Black Tree in detail. 6M
3. Write properties of Red-Black Tree. 3M
4. Construct Red-Black Tree for given numbers:

   10,18,7,15,16,20,25,40,60,21,70. 8M

Dr Ratna Patil

Thank You…!

Dr Ratna Patil