DATA STRUCTURE ALGORITHMS AND APPLICATIONS (CT- 159)

Lecture – 10
AVL Tree

Instructor: Engr. Nasr Kamal

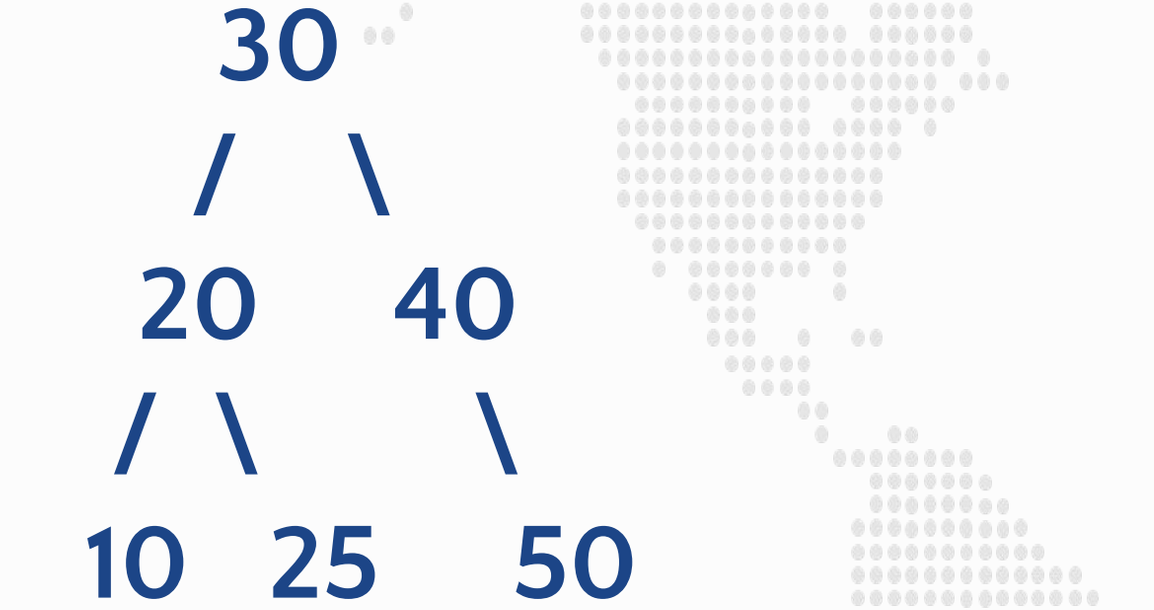
Department of Computer Science and Information Technology

AVL Tree

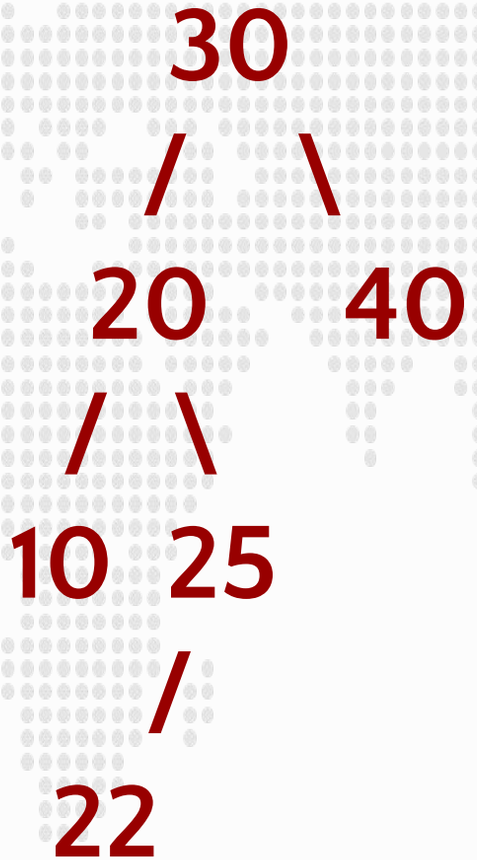
- AVL tree has been named after two persons Adelson-Velskii and Landis. These two had devised a technique to make the tree balanced. According to them:

> An **AVL Tree** is a type of self-balancing Binary Search Tree (BST) where the difference in heights between the left and right subtrees of every node (known as the **balance factor**) is at most 1.
> This balance ensures that the tree remains roughly balanced, resulting in better search, insertion, and deletion times—guaranteed to be O(logn).

AVL Tree

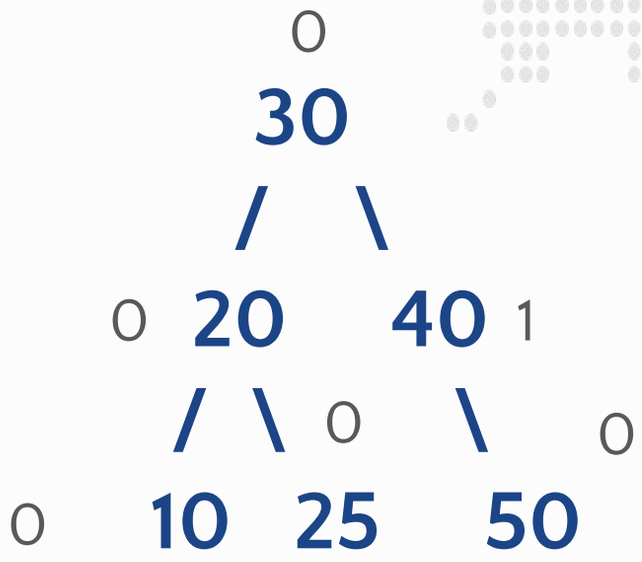


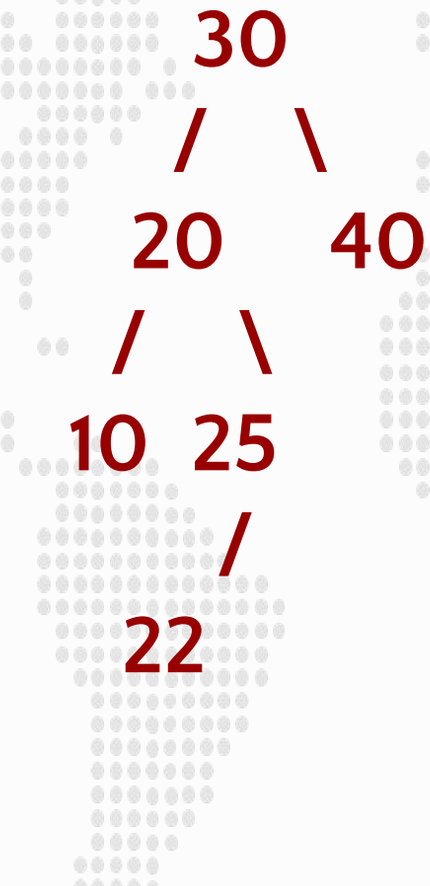
Balanced Tree (AVL Tree)

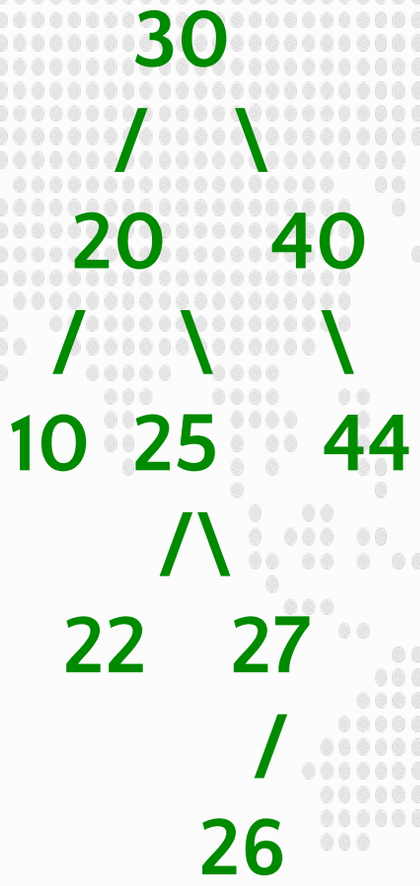


Not an AVL

Let's find out the BF in AVL Tree







Properties of AVL Tree

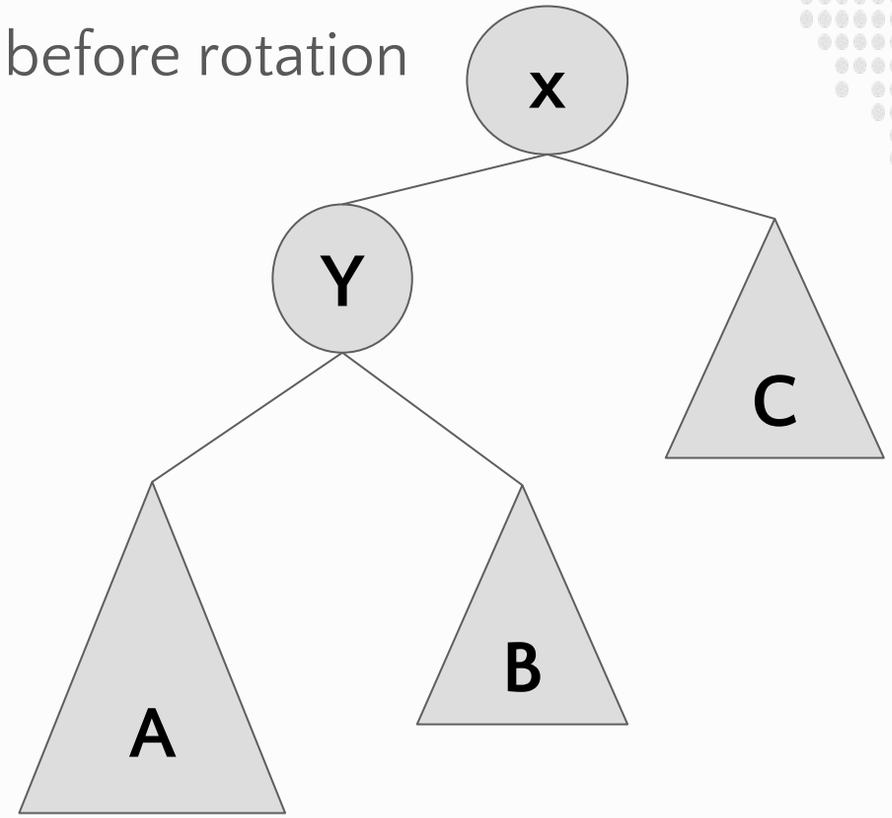
- **Balance Factor**: The balance factor of a node is the height difference between its left and right subtrees.

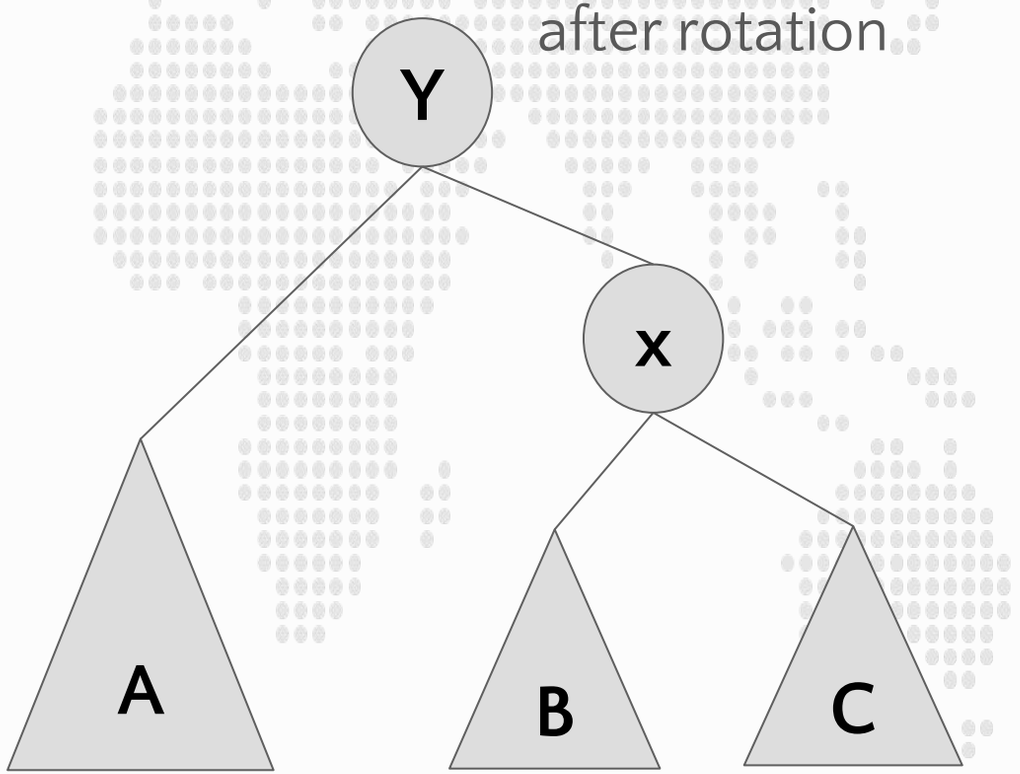
  Balance Factor=Height of Left Subtree−Height of Right

- A node is considered balanced if its balance factor is −1, 0, or 1.
- If inserting or deleting a node causes the balance factor of any node to exceed 1 or −1, rotations are used to restore balance.

Rotation

- When the tree structure changes (e.g., insertion or deletion), we need to transform the tree to restore the AVL tree property.
- This is done using single rotations or double rotations.



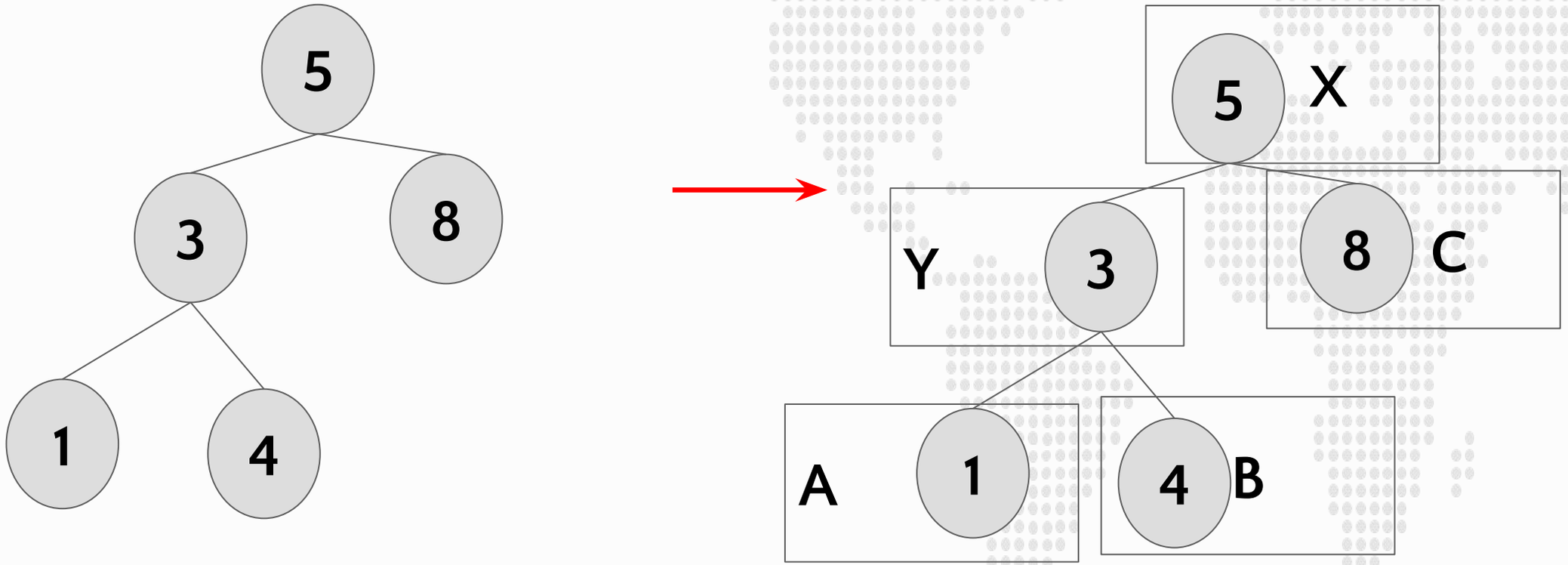


Rotation

- Since an insertion/deletion involves adding/deleting a single node, this can only increase/decrease the height of some subtree by 1
- Thus, 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.
- Rotations will be applied to x to restore the AVL tree property.
- There are four types of rotations:
    - ➢ Right Rotation
    - ➢ Left Rotation
    - ➢ Left–Right Rotation
    - ➢ Right–Left Rotation

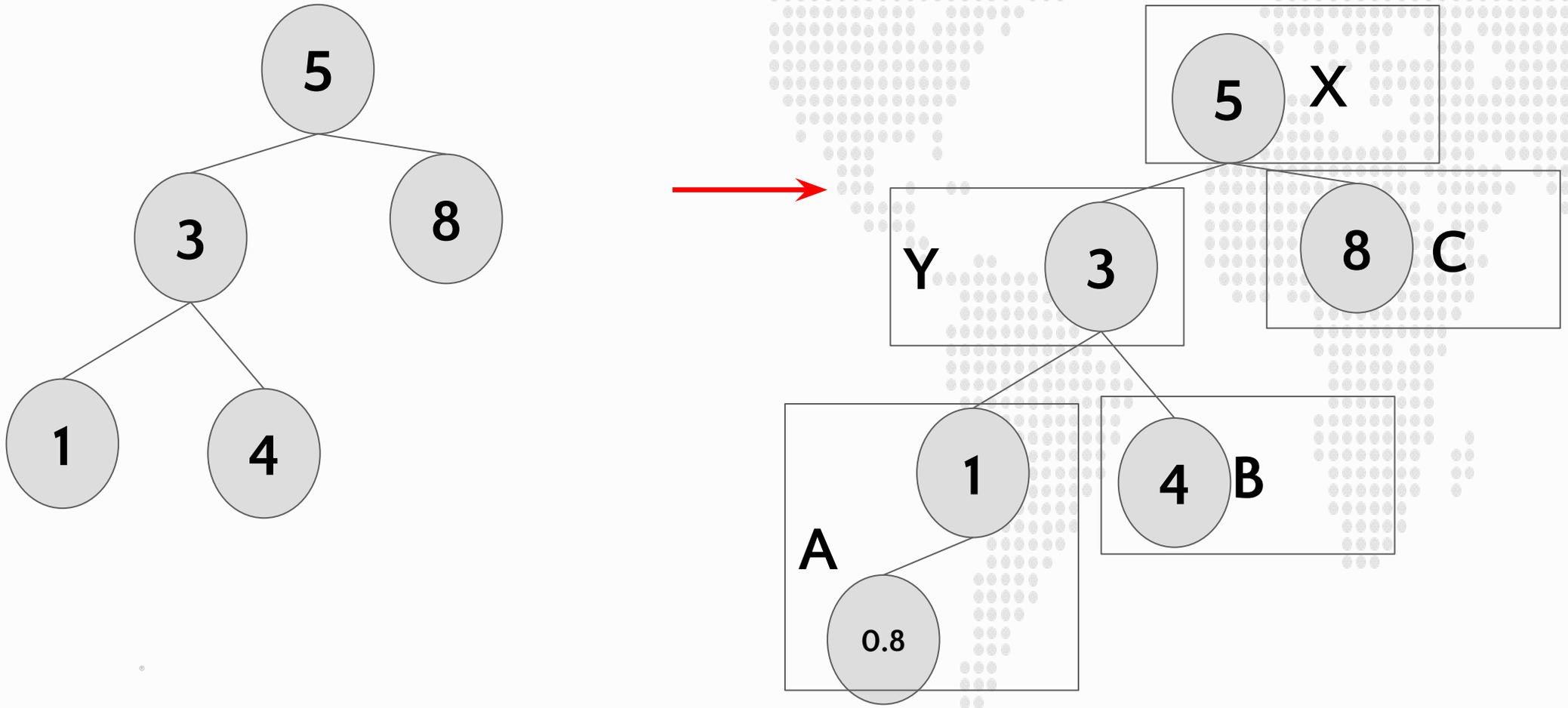
Single Right Rotation (Left-Left Case)

Rotation is performed when a node becomes unbalanced due to too much height on its left subtree

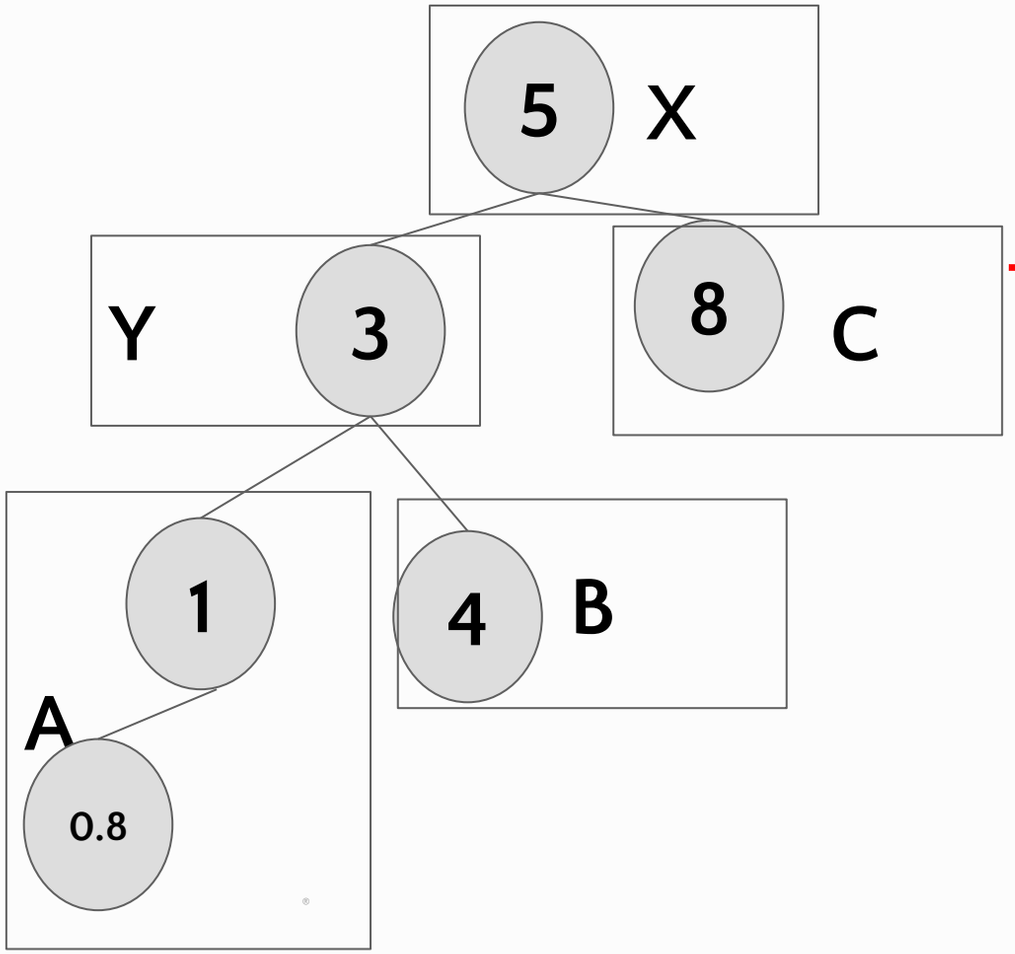


Single Right Rotation (Left–Left Case)

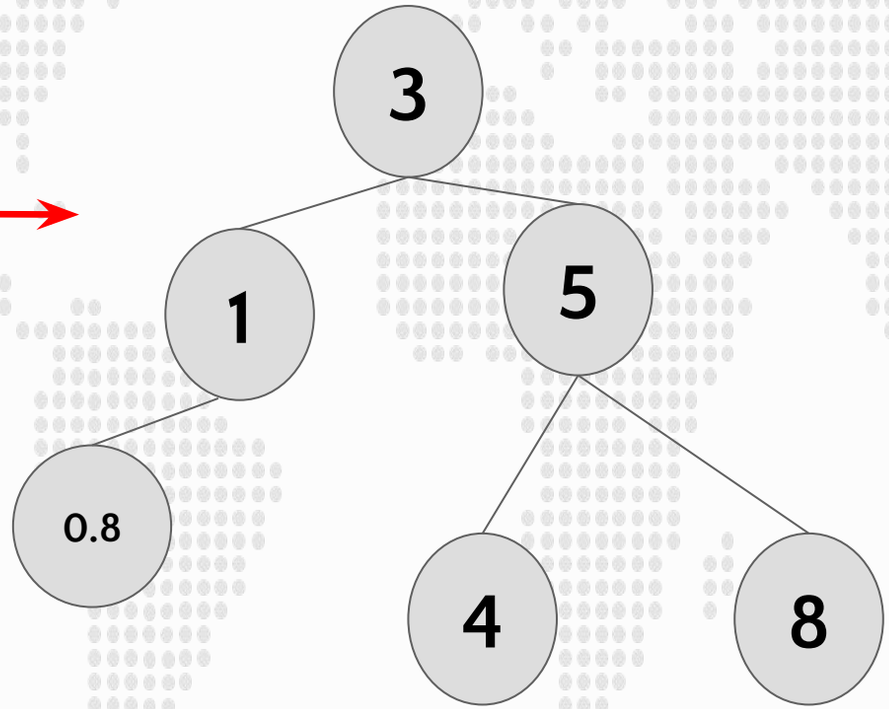
Insert 0.8



Single Right Rotation (Left–Left Case)

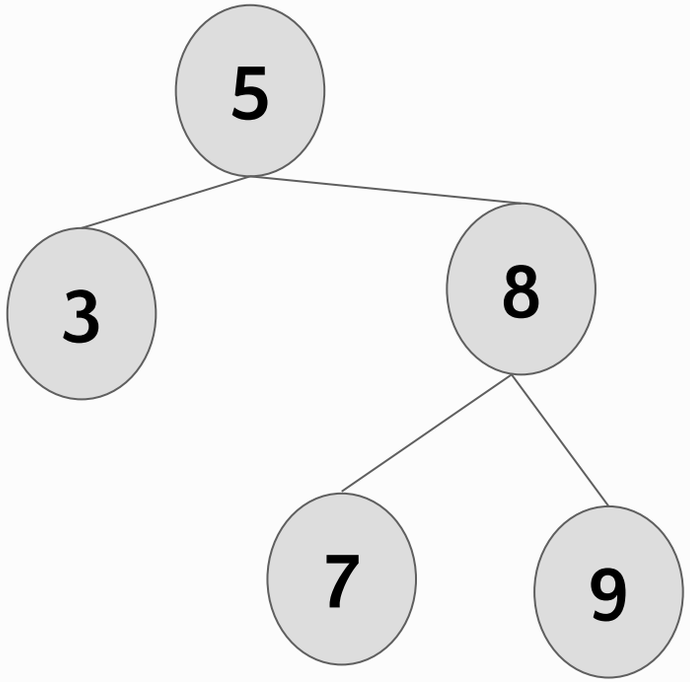


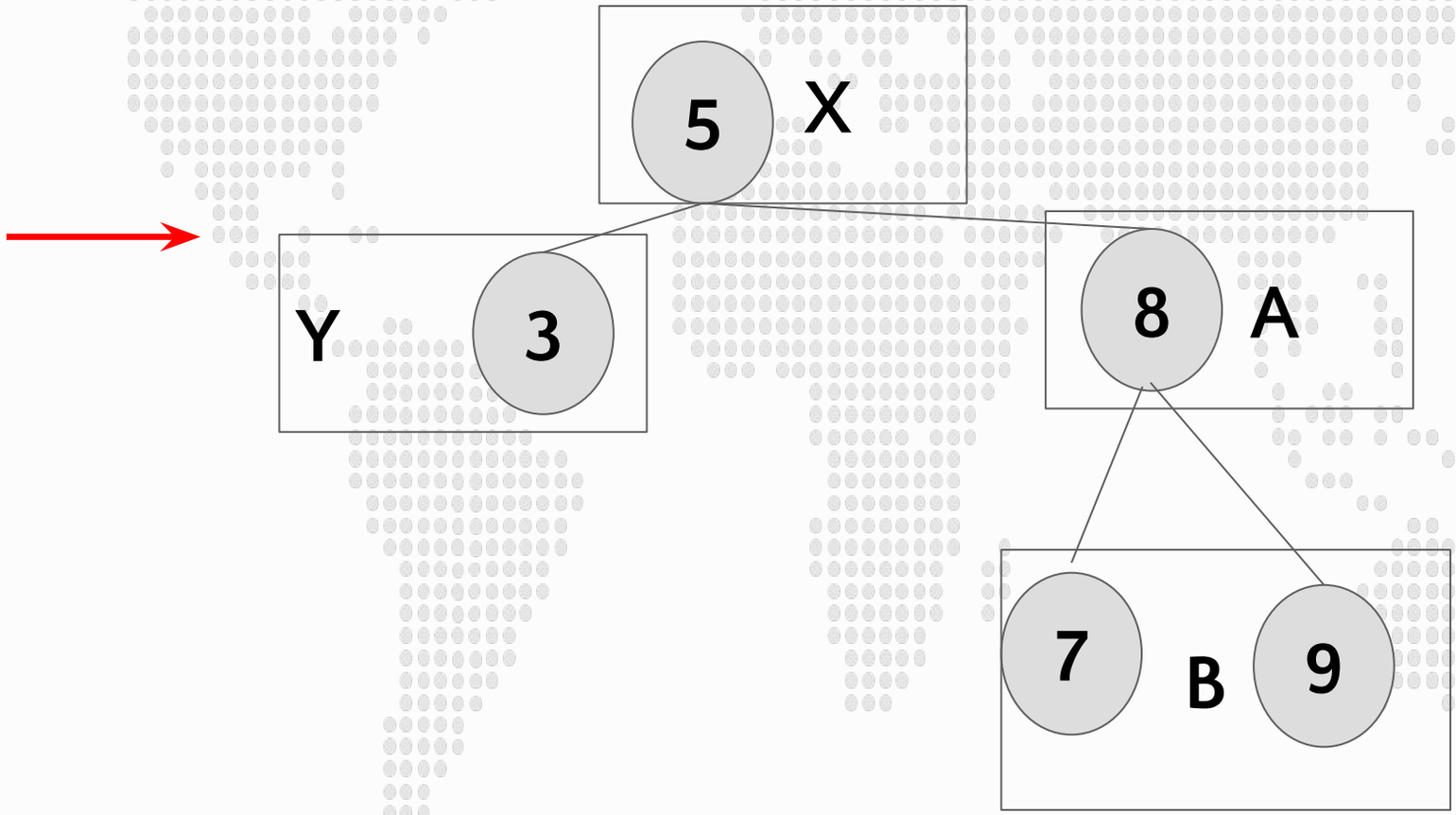
After Rotation



Single Left Rotation (Right–Right Case)

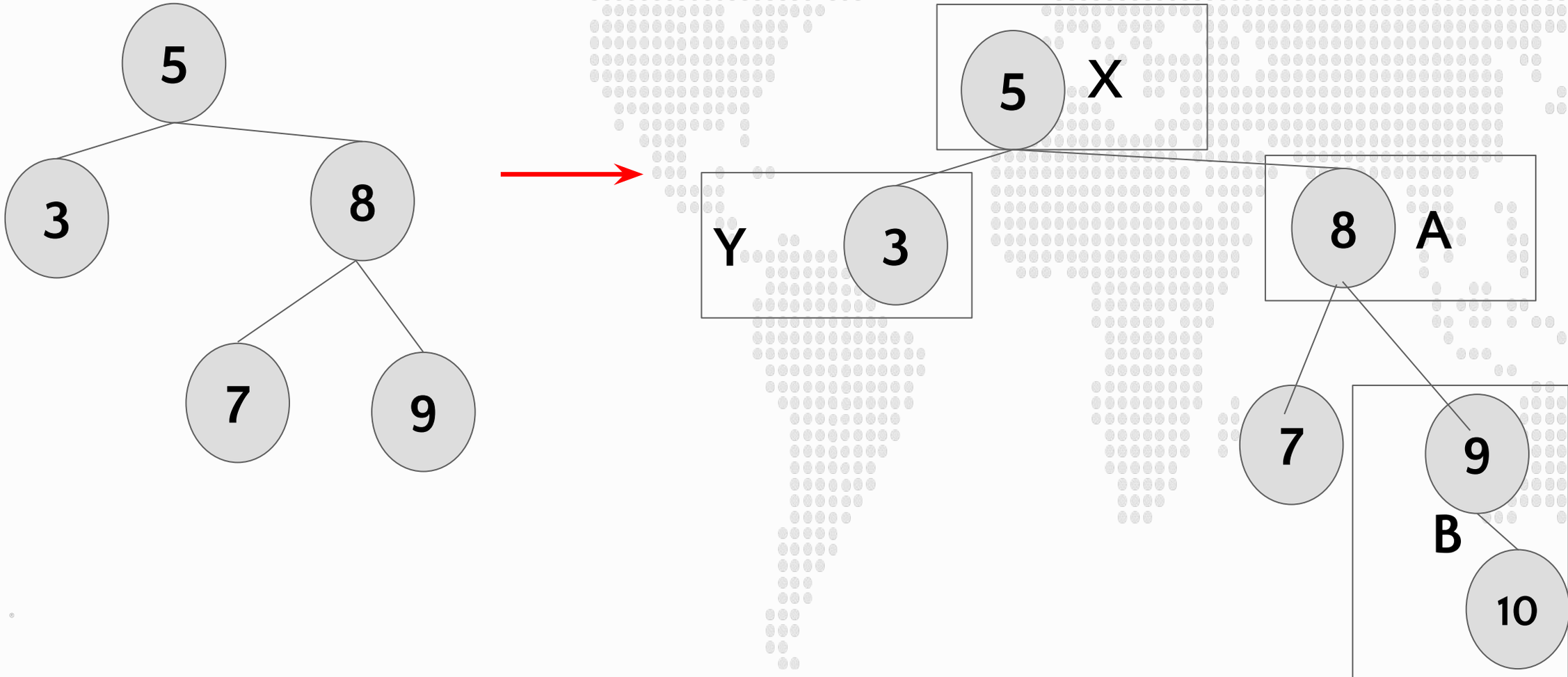
Performed when a node becomes unbalanced due to too much height on its right subtree



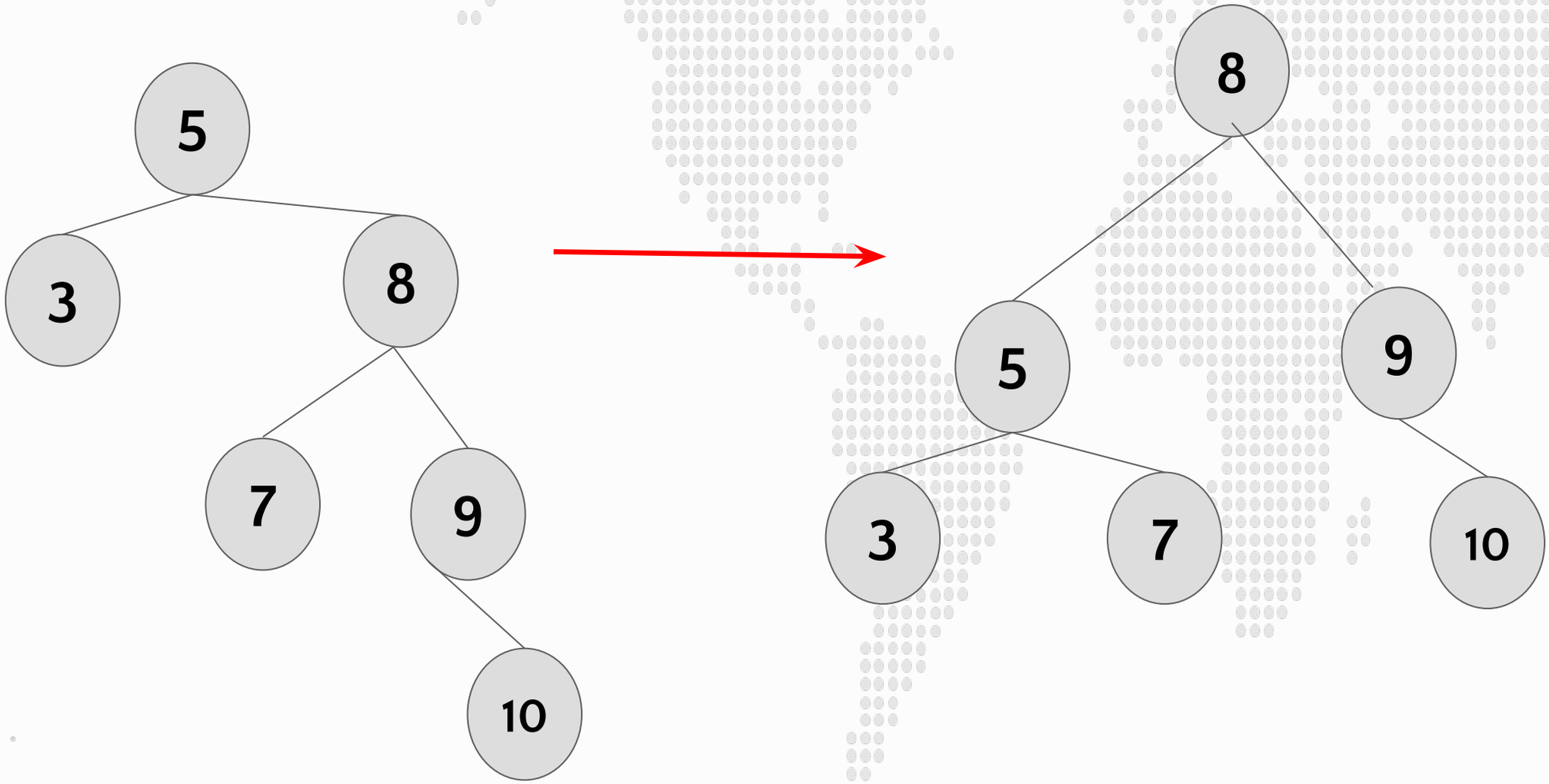


Single Left Rotation (Right–Right Case)

Insert 10

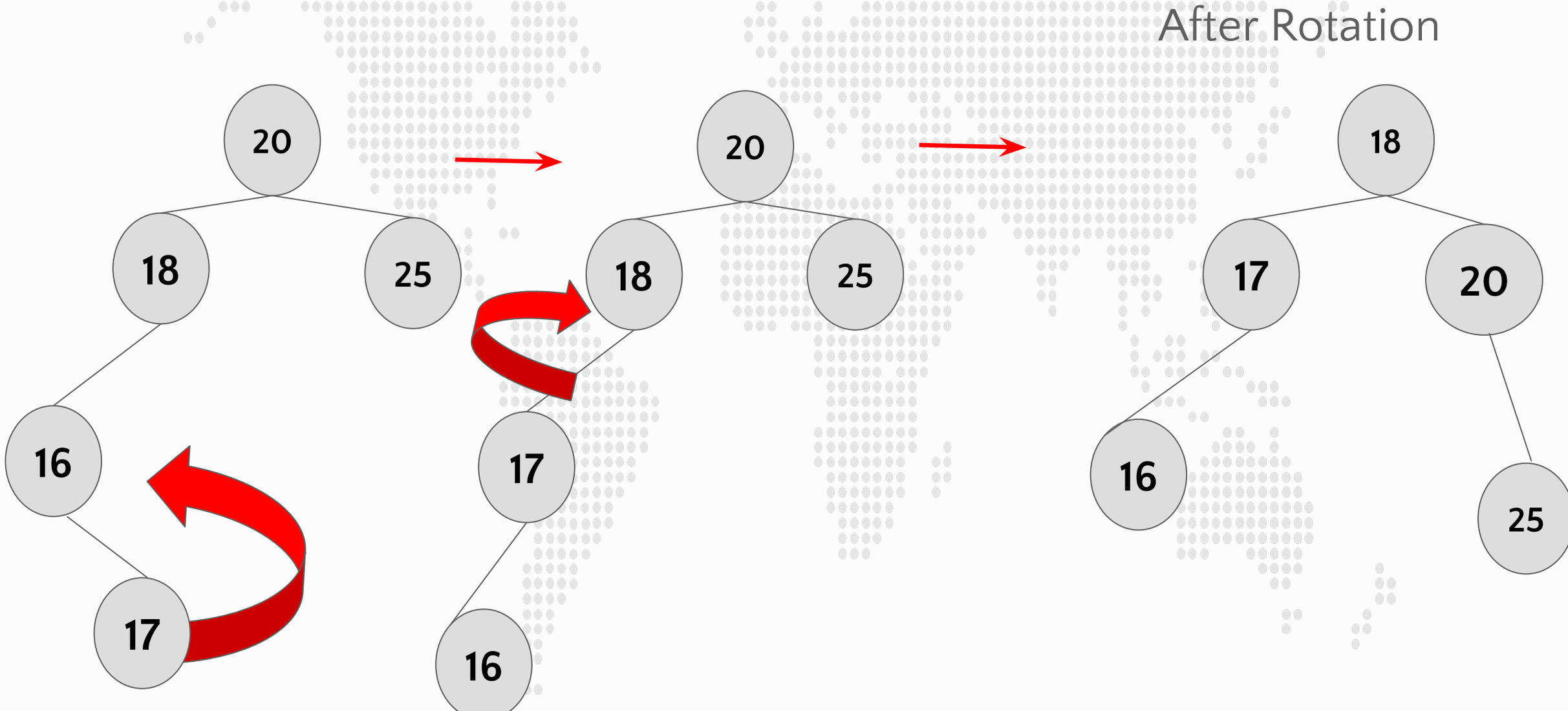


Single Left Rotation (Right–Right Case)



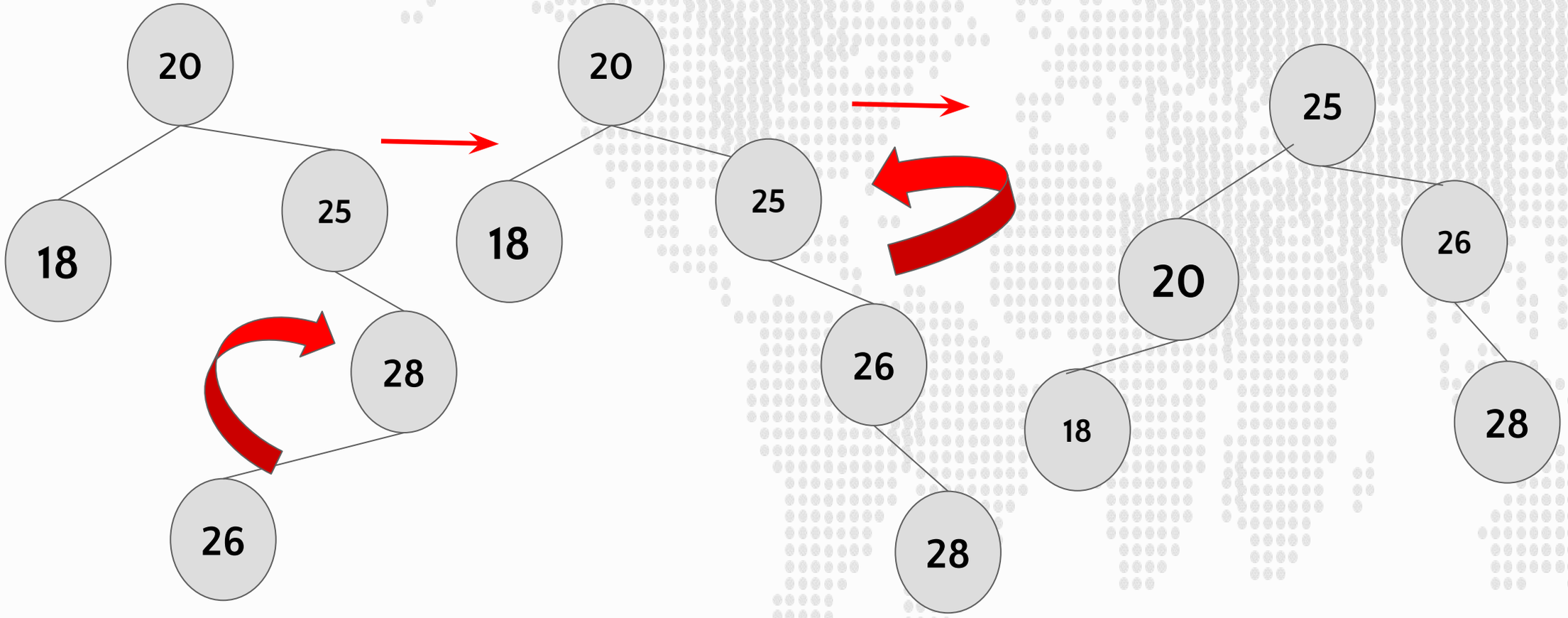
Left-Right Rotation

Performed when a node's left child is right-heavy (left-right case).



Right–Left Rotation

Performed when a node's right child is left-heavy (right–left case).



Steps for Inserting a Node in an AVL tree

1. Perform standard BST insertion.
2. Update the height of each node.
3. Calculate the balance factor for each node.
4. If unbalanced, apply the appropriate rotation(s)

Steps for Deleting a Node from an AVL tree

1. Perform Standard BST Deletion.
2. Update the height for each node.
3. Check Balance Factor for each node.
4. Perform Rotations (if needed). If the tree is unbalanced, apply one of the four rotations: LL, RR, LR, or RL to restore balance.
5. Repeat Until the Tree is Fully Balanced. Ensure that every node along the path from the deleted node up to the root is balanced. Multiple rotations might be needed.

An Extended Example

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

Fig 1

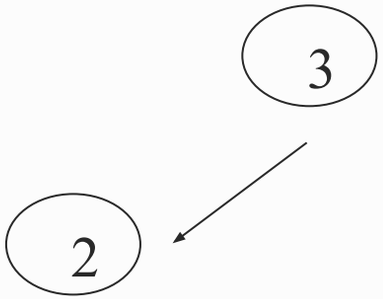


Fig 2

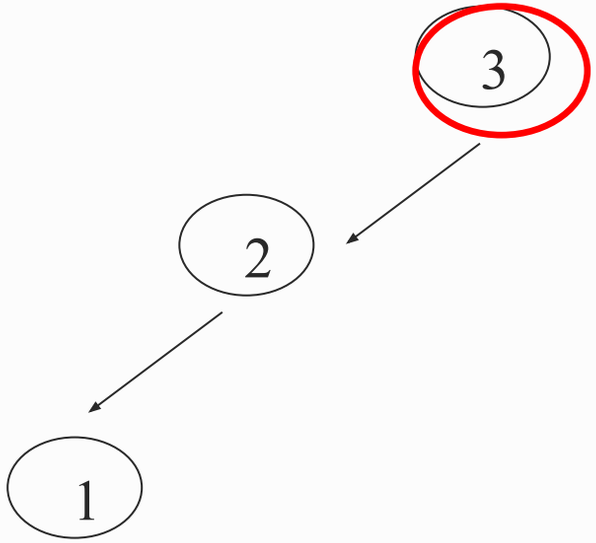


Fig 3

Single rotation

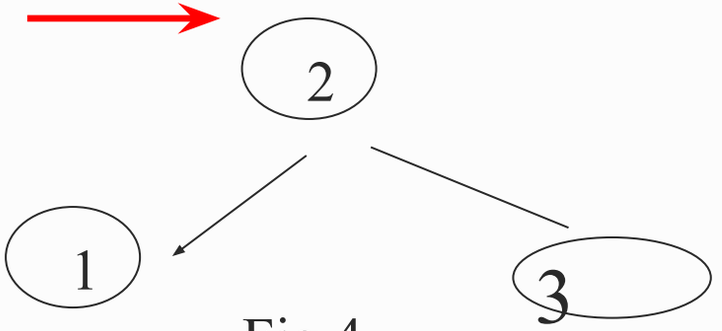


Fig 4

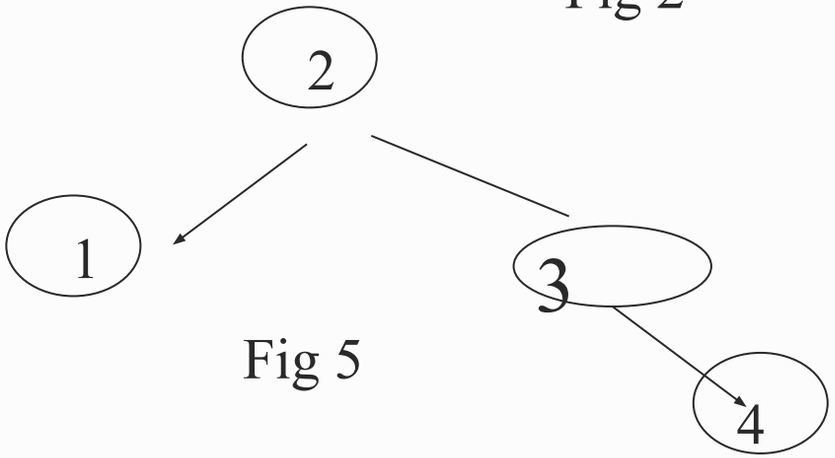


Fig 5

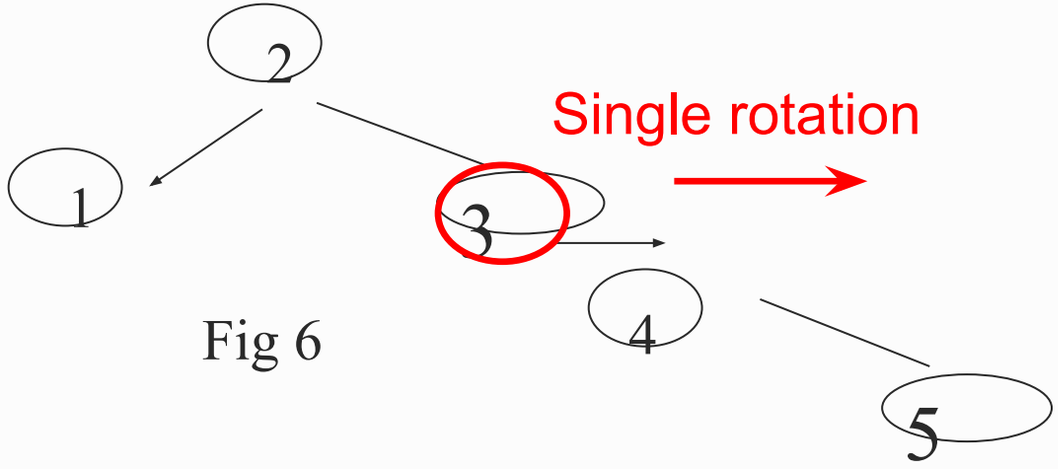


Fig 6

Single rotation

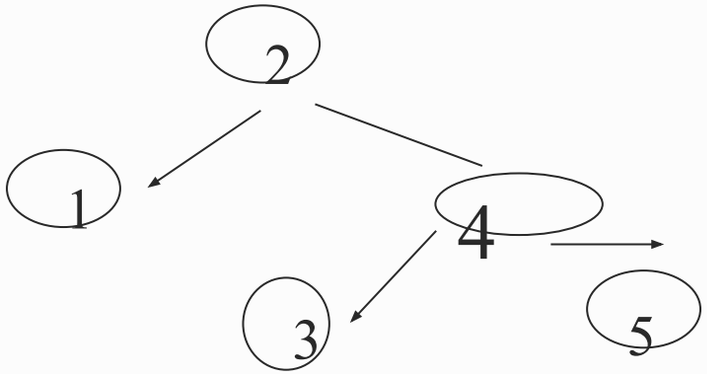
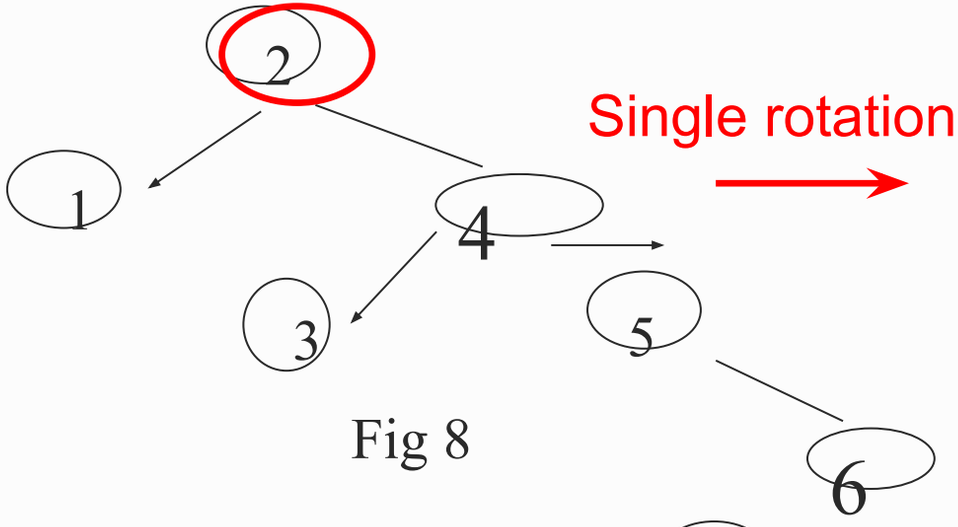


Fig 7



Single rotation

Fig 8

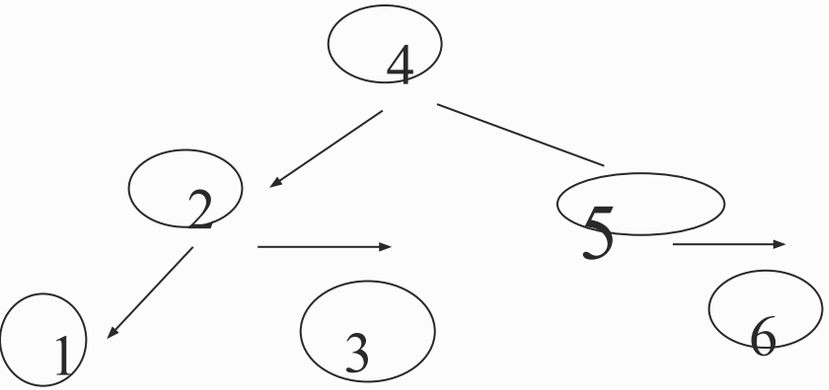
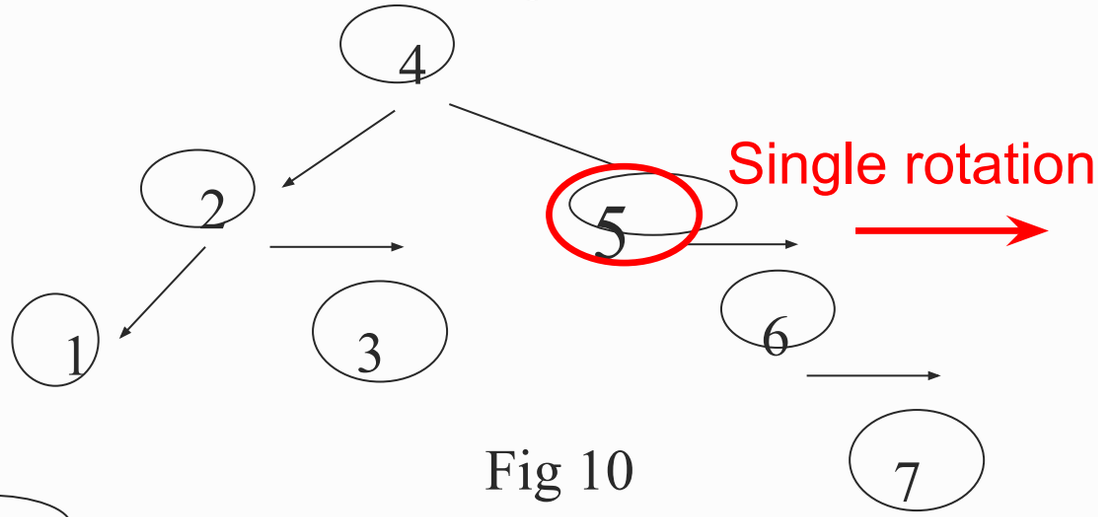


Fig 9



Single rotation

Fig 10

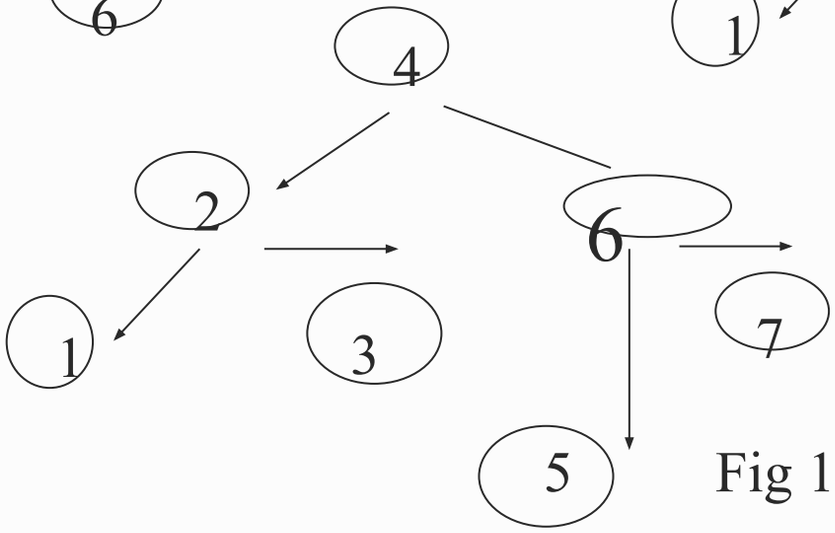


Fig 11

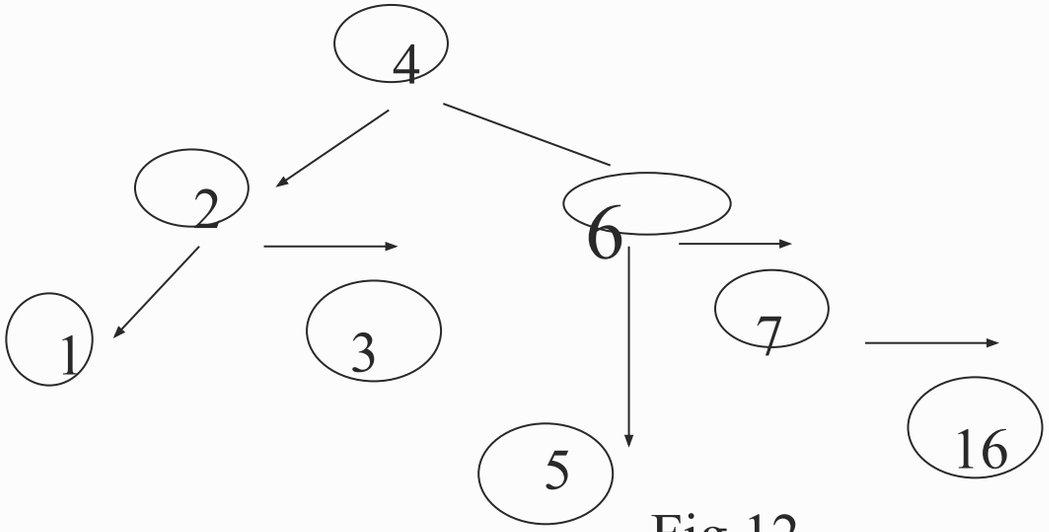


Fig 12

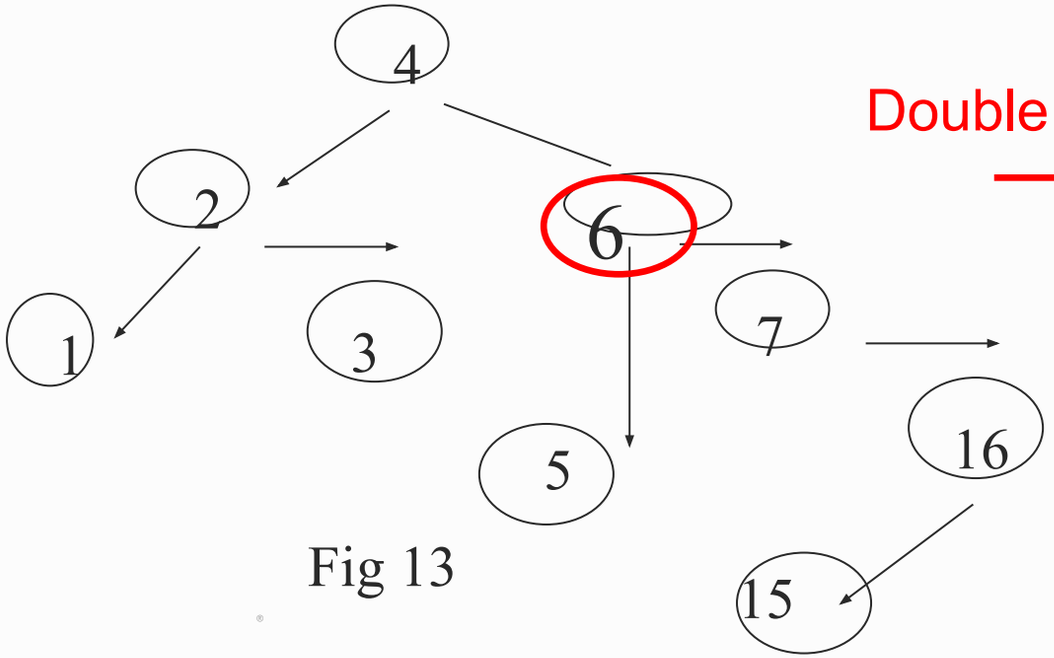


Fig 13

Double rotation

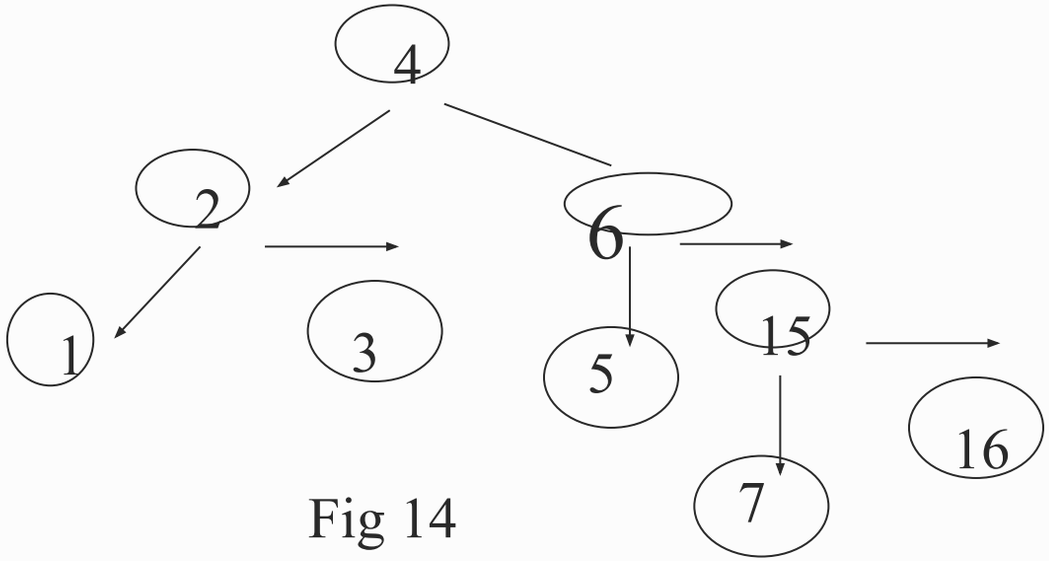


Fig 14

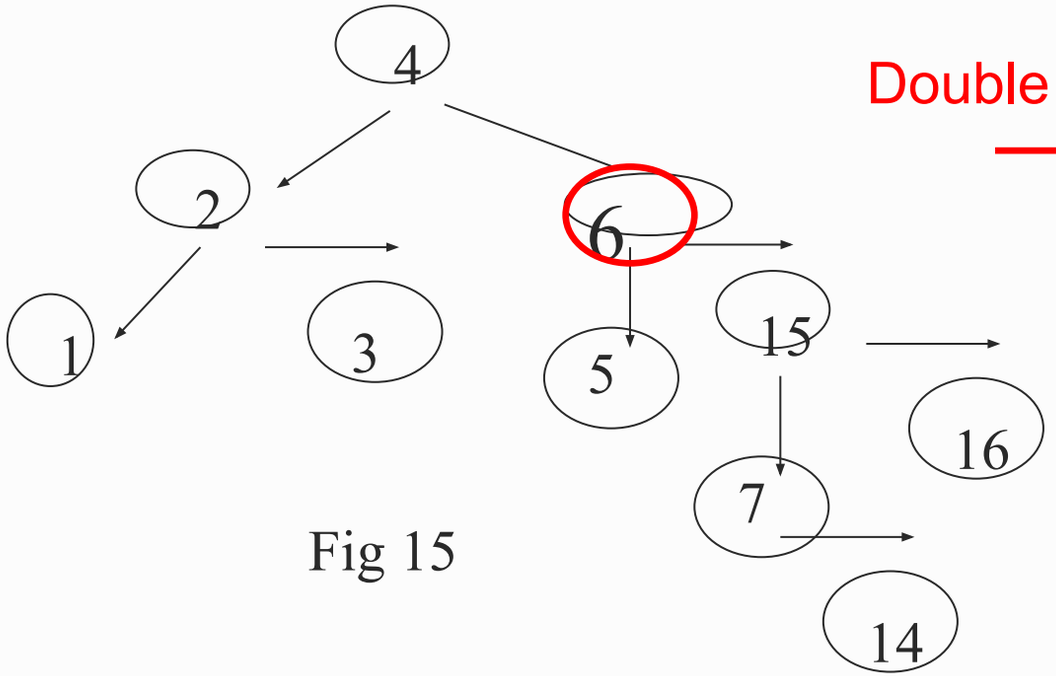


Fig 15

Double rotation

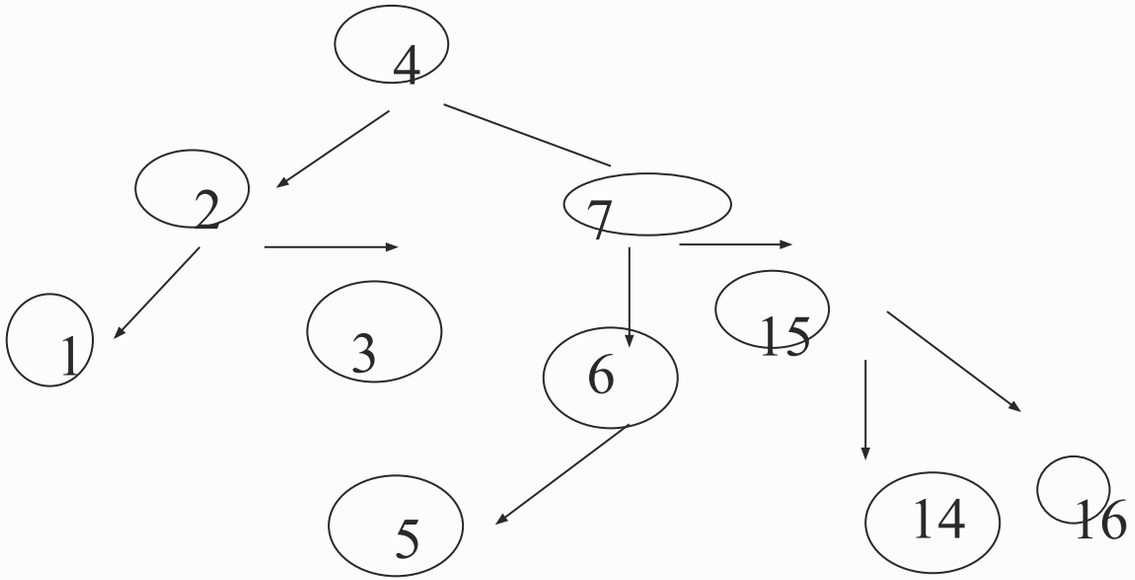


Fig 16