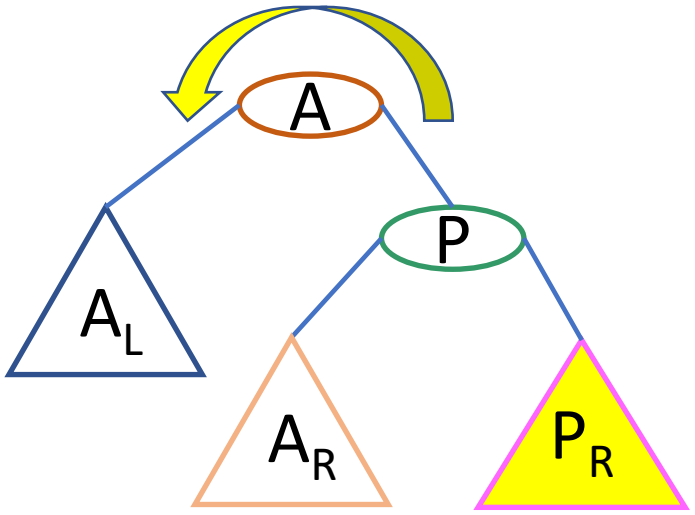
CS2x1:Data Structures and Algorithms

Koteswararao Kondepu

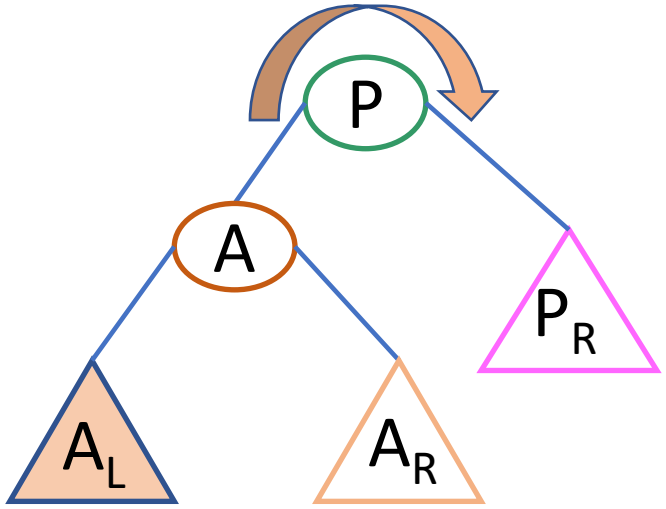
k.kondepu@iitdh.ac.in

Recap: AVL Tree

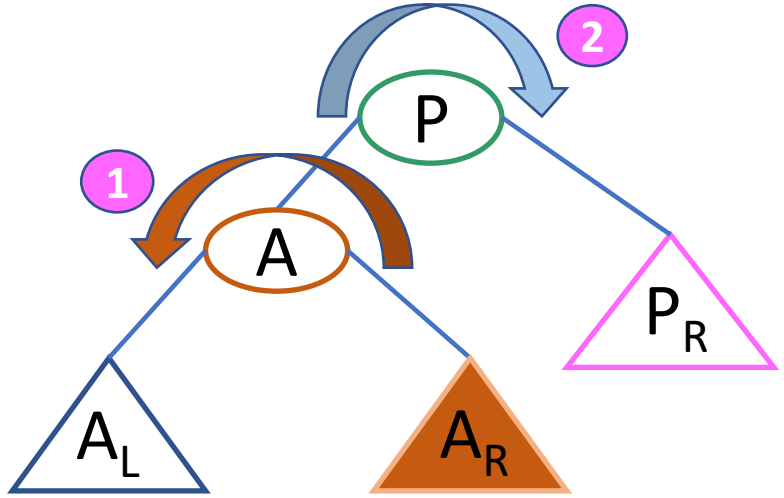
"***Left Rotation***"



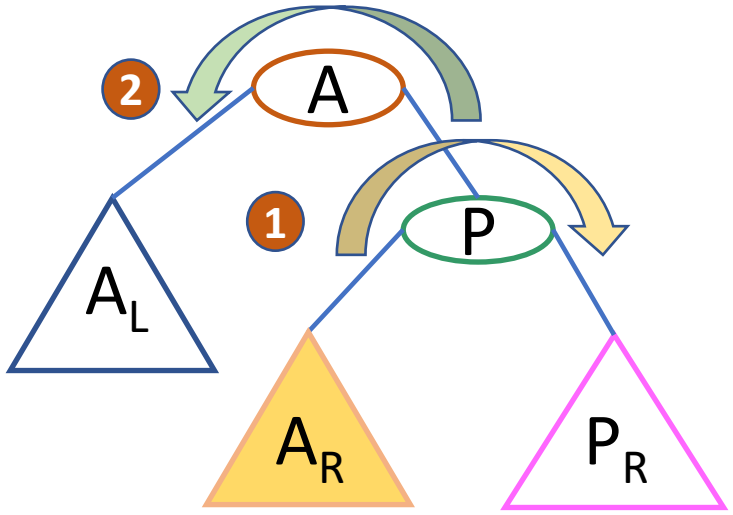
"***Right Rotation***"



"***Left Right Rotation***"



"***Right Left Rotation***"



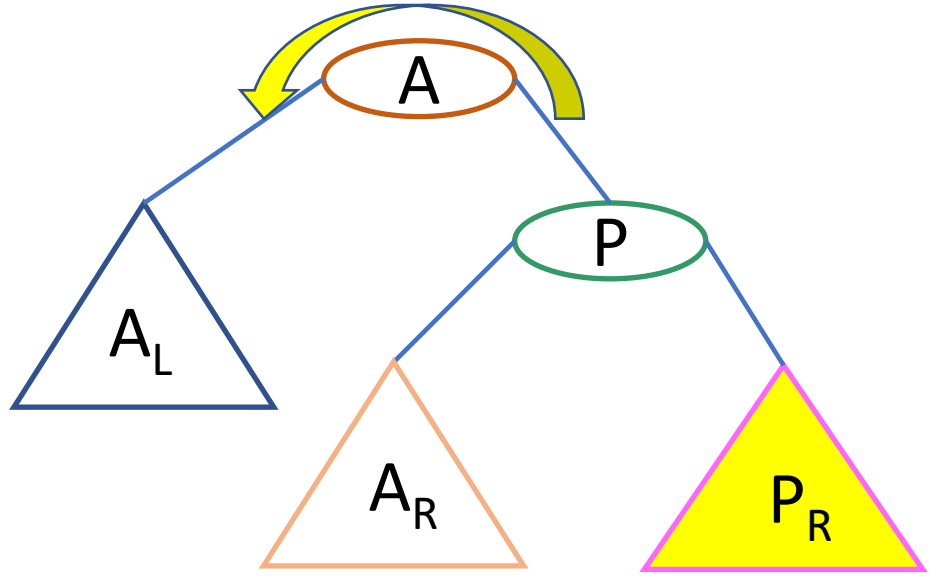
1 Left Rotation 2 Right Rotation

1 Right Rotation 2 Left Rotation

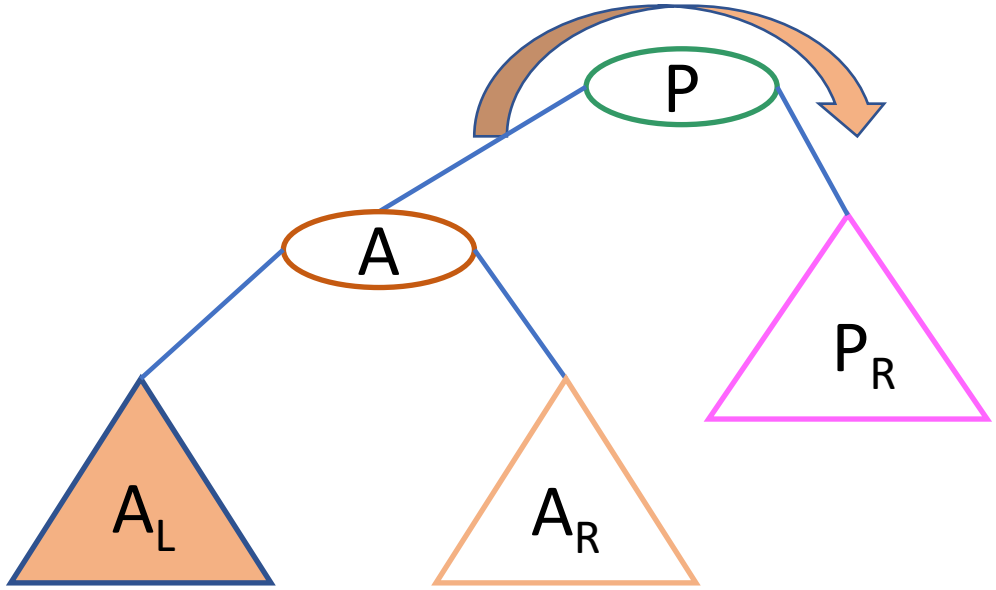
AVL Tree: Construction

72, 73, 74, 66, 65, 69, 67, 70, 68, 71, 75, 76

"***Left Rotation***"

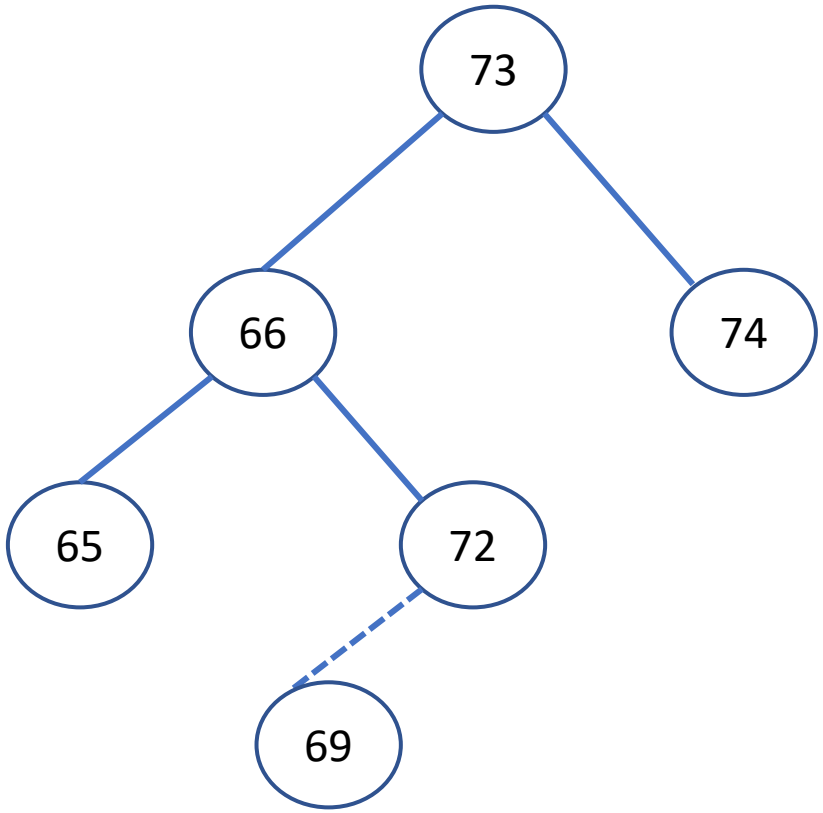


"***Right Rotation***"

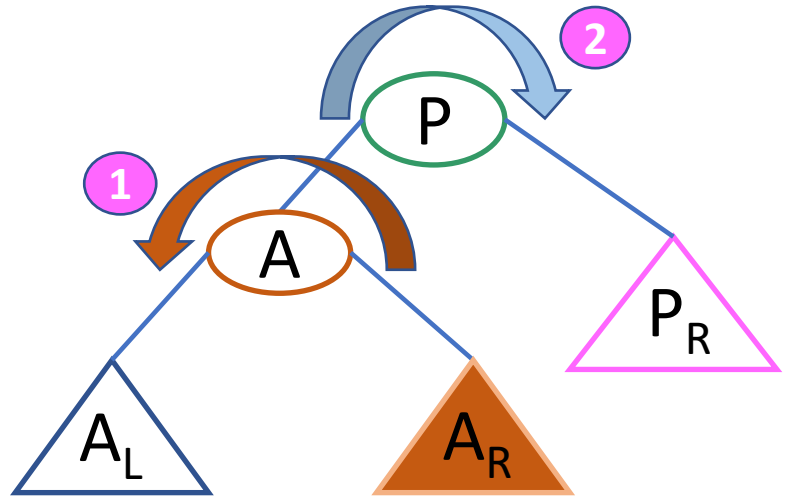


AVL Tree: Construction

72, 73, 74, 66, 65, 69, 67, 70, 68, 71, 75, 76



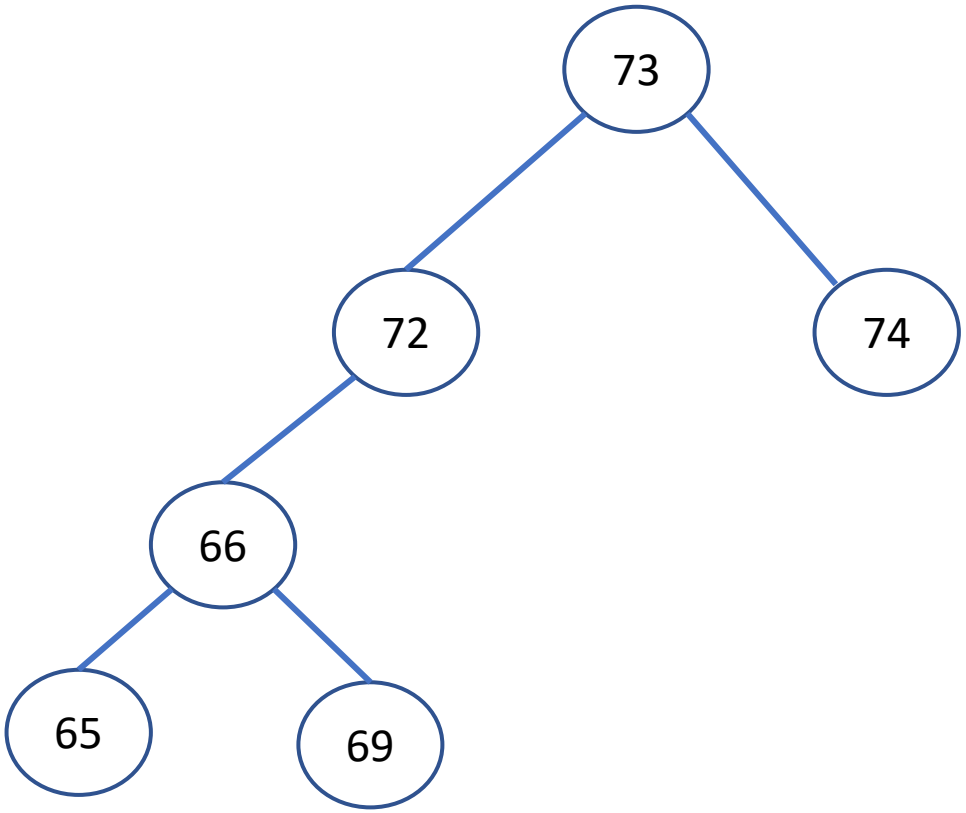
"***Left Right Rotation***"



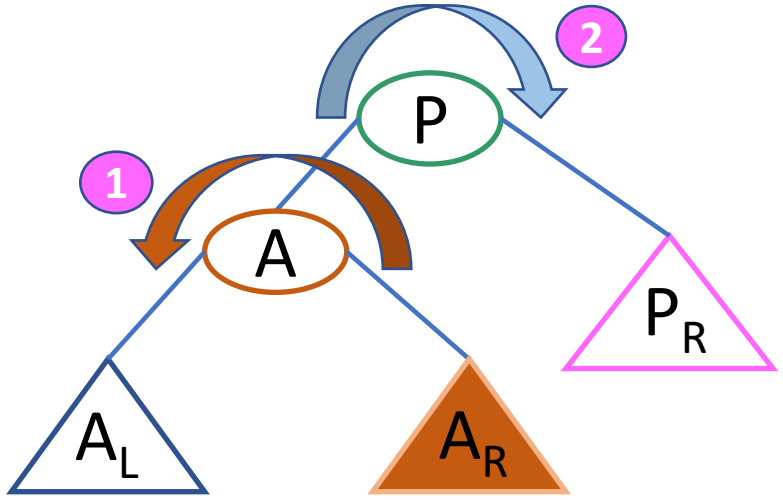
1 Left Rotation 2 Right Rotation

AVL Tree: Construction

72, 73, 74, 66, 65, 69, 67, 70, 68, 71, 75, 76



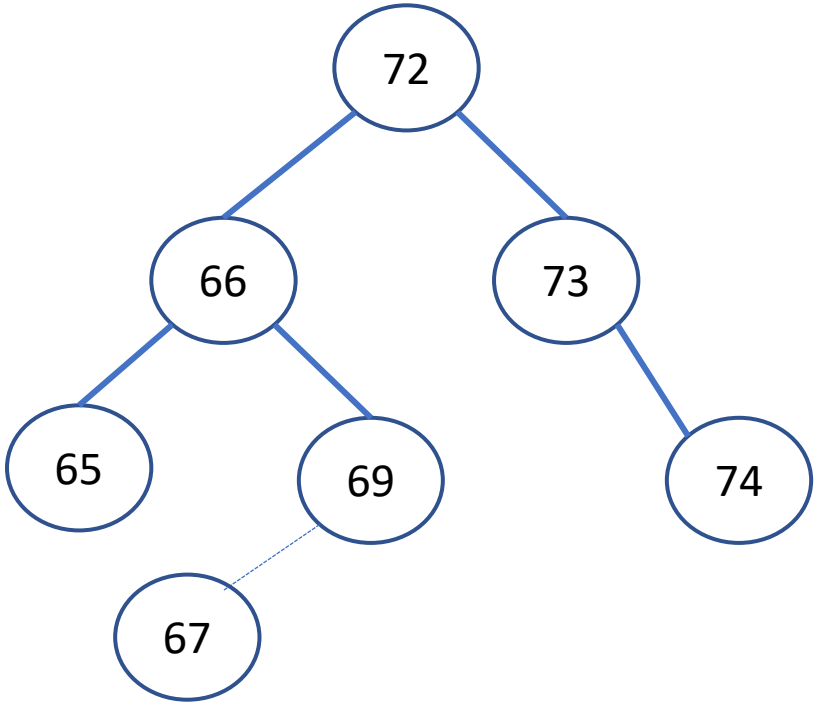
"***Left Right Rotation***"

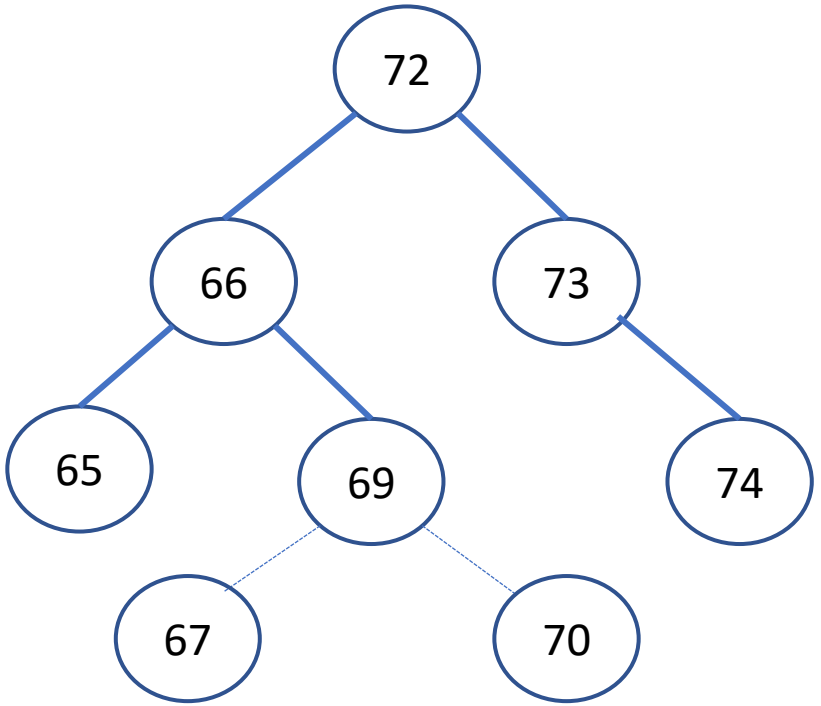


1 Left Rotation 2 Right Rotation

AVL Tree: Construction

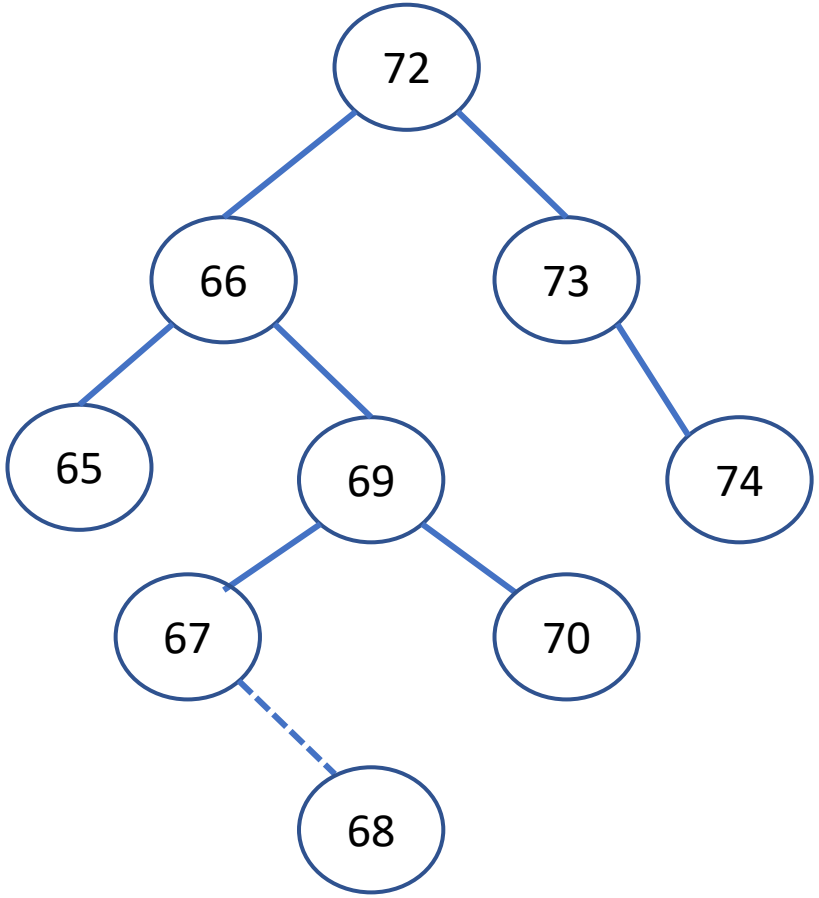
72, 73, 74, 66, 65, 69, 67, 70, 68, 71, 75, 76



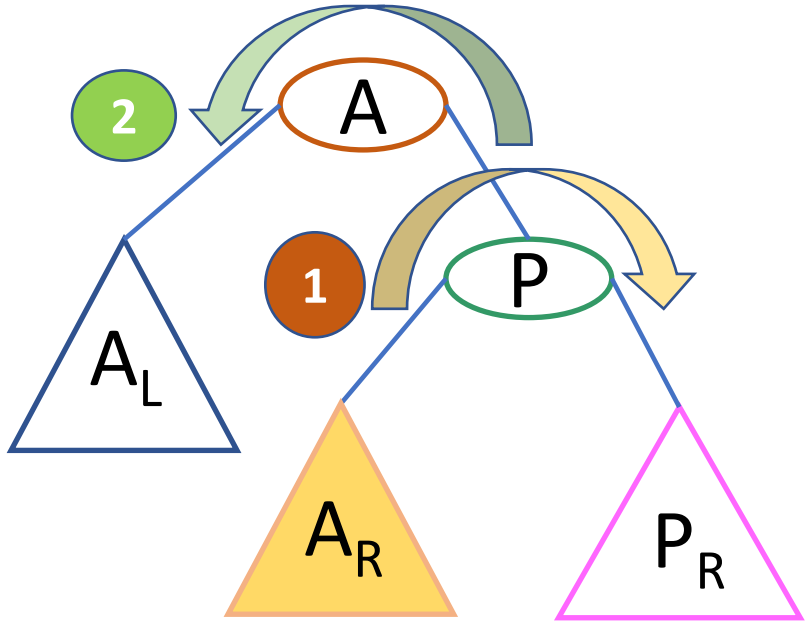


AVL Tree: Construction

72, 73, 74, 66, 65, 69, 67, 70, 68, 71, 75, 76



"***Right Left Rotation***"

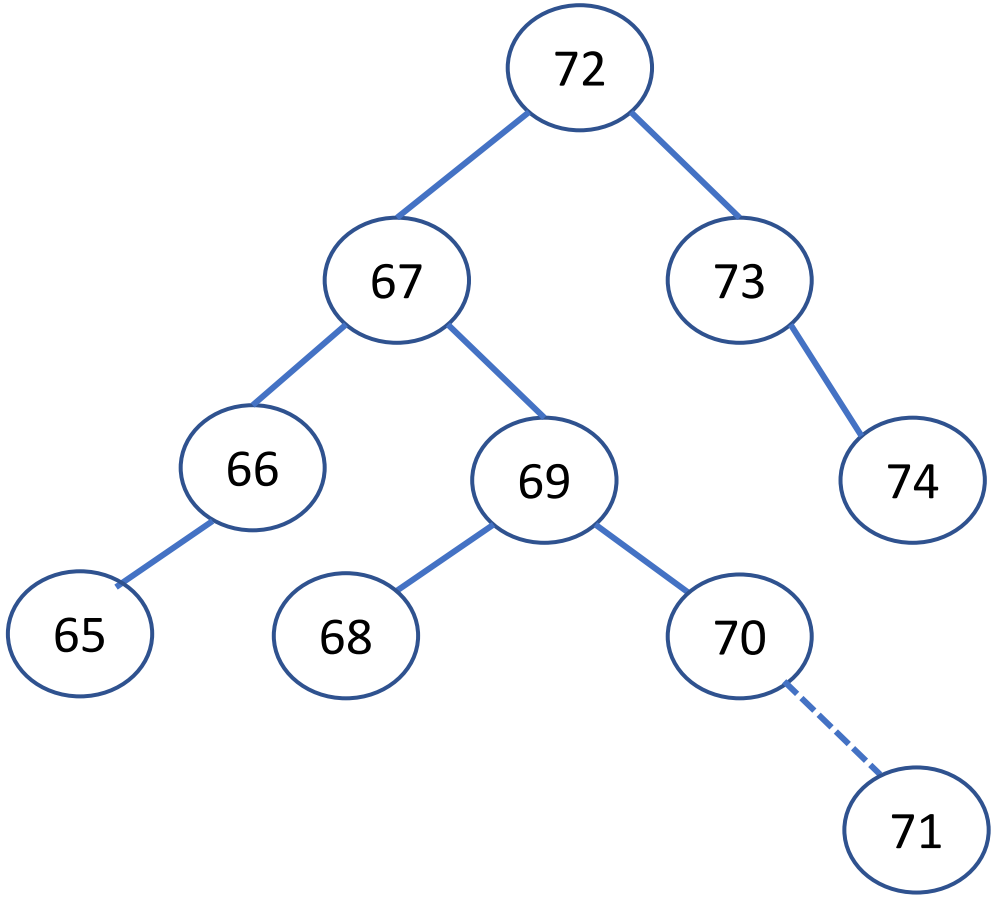


1 Right Rotation

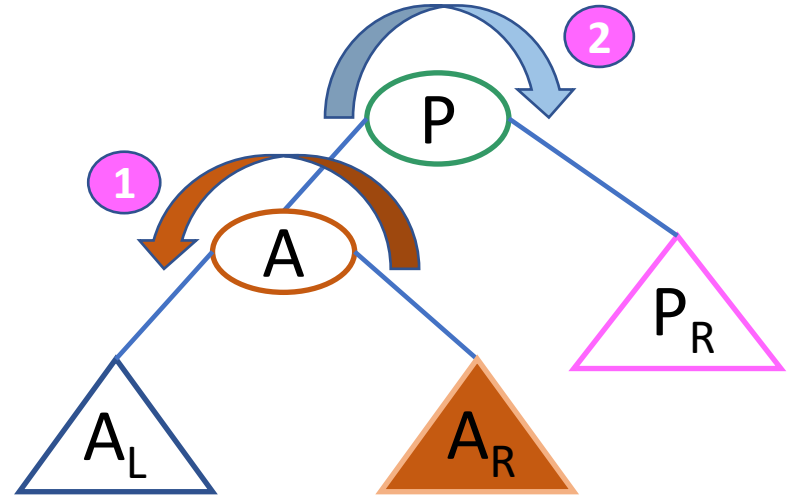
2 Left Rotation

AVL Tree: Construction

72, 73, 74, 66, 65, 69, 67, 70, 68, 71, 75, 76



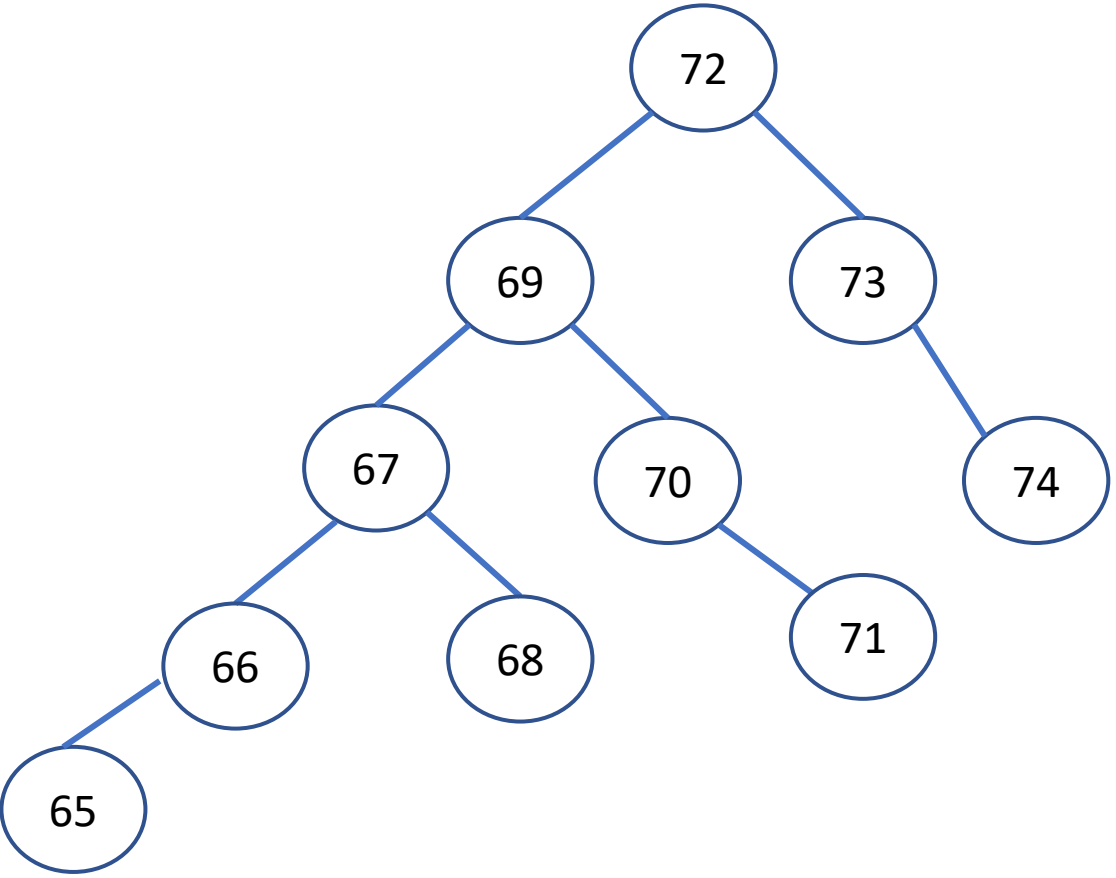
"***Left Right Rotation***"



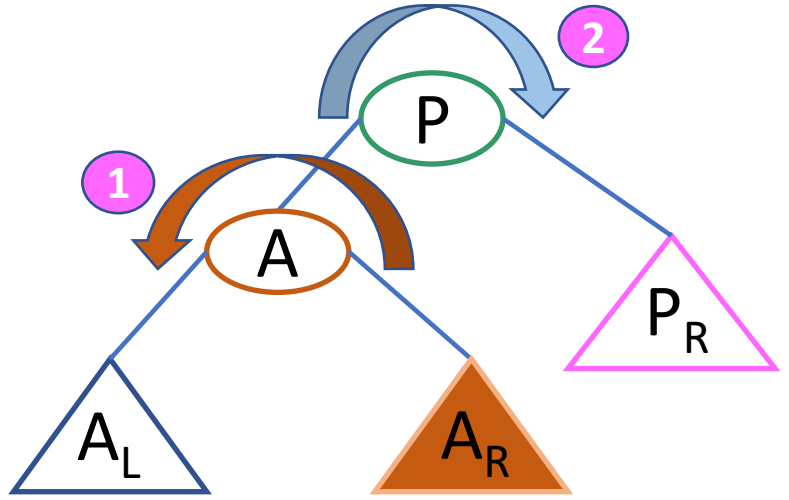
1 Left Rotation 2 Right Rotation

AVL Tree: Construction

72, 73, 74, 66, 65, 69, 67, 70, 68, 71, 75, 76



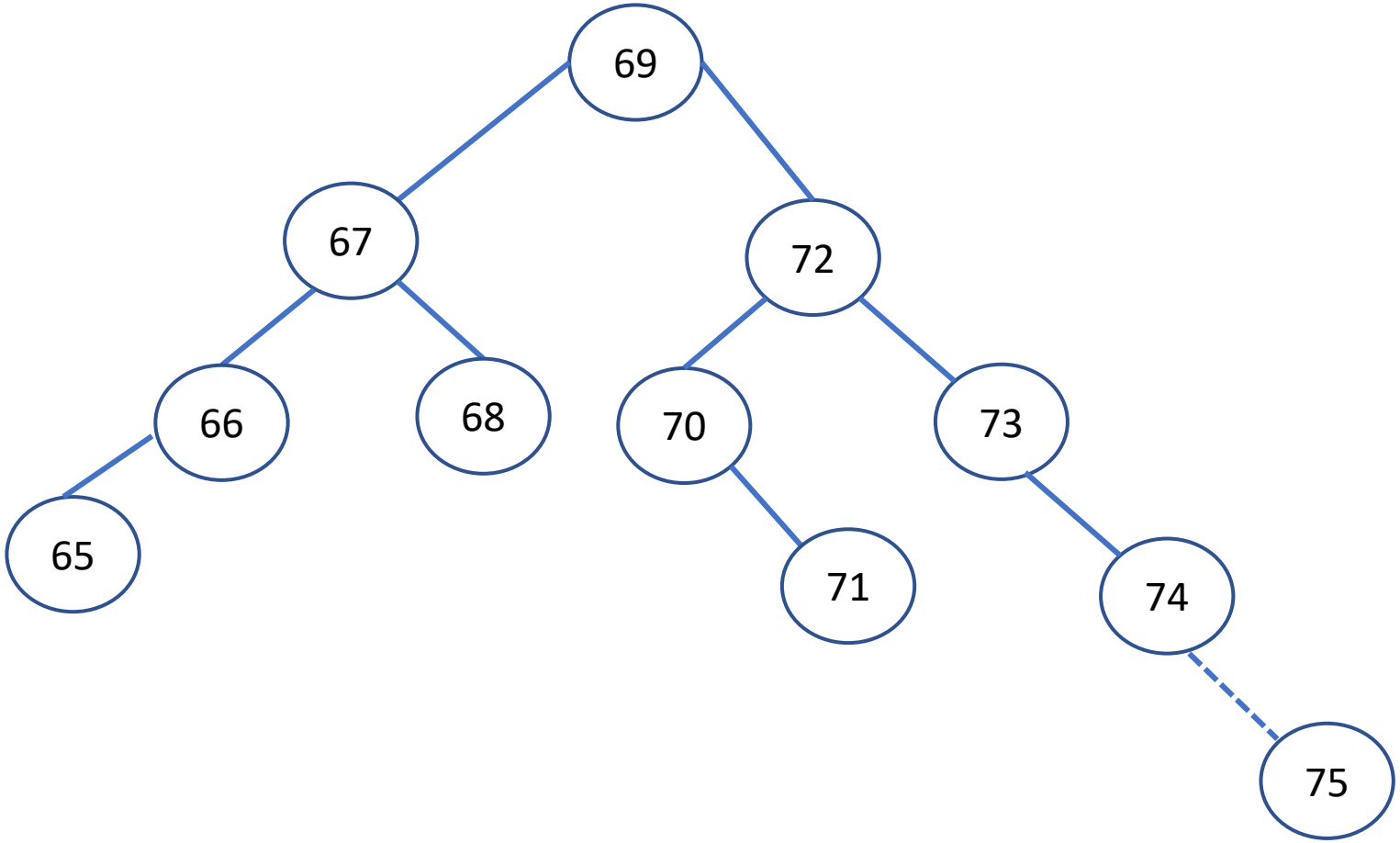
"***Left Right Rotation***"



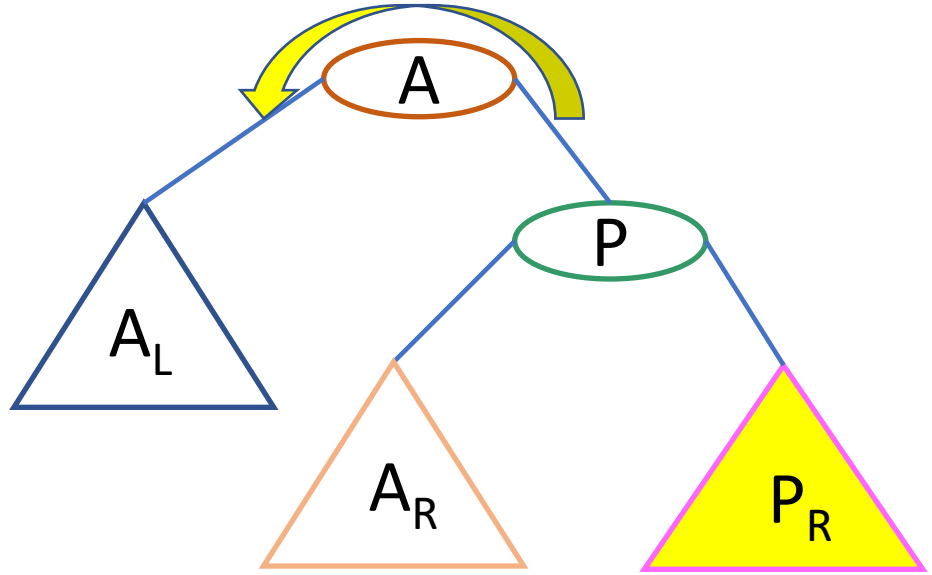
1 Left Rotation 2 Right Rotation

AVL Tree: Construction

72, 73, 74, 66, 65, 69, 67, 70, 68, 71, 75, 76

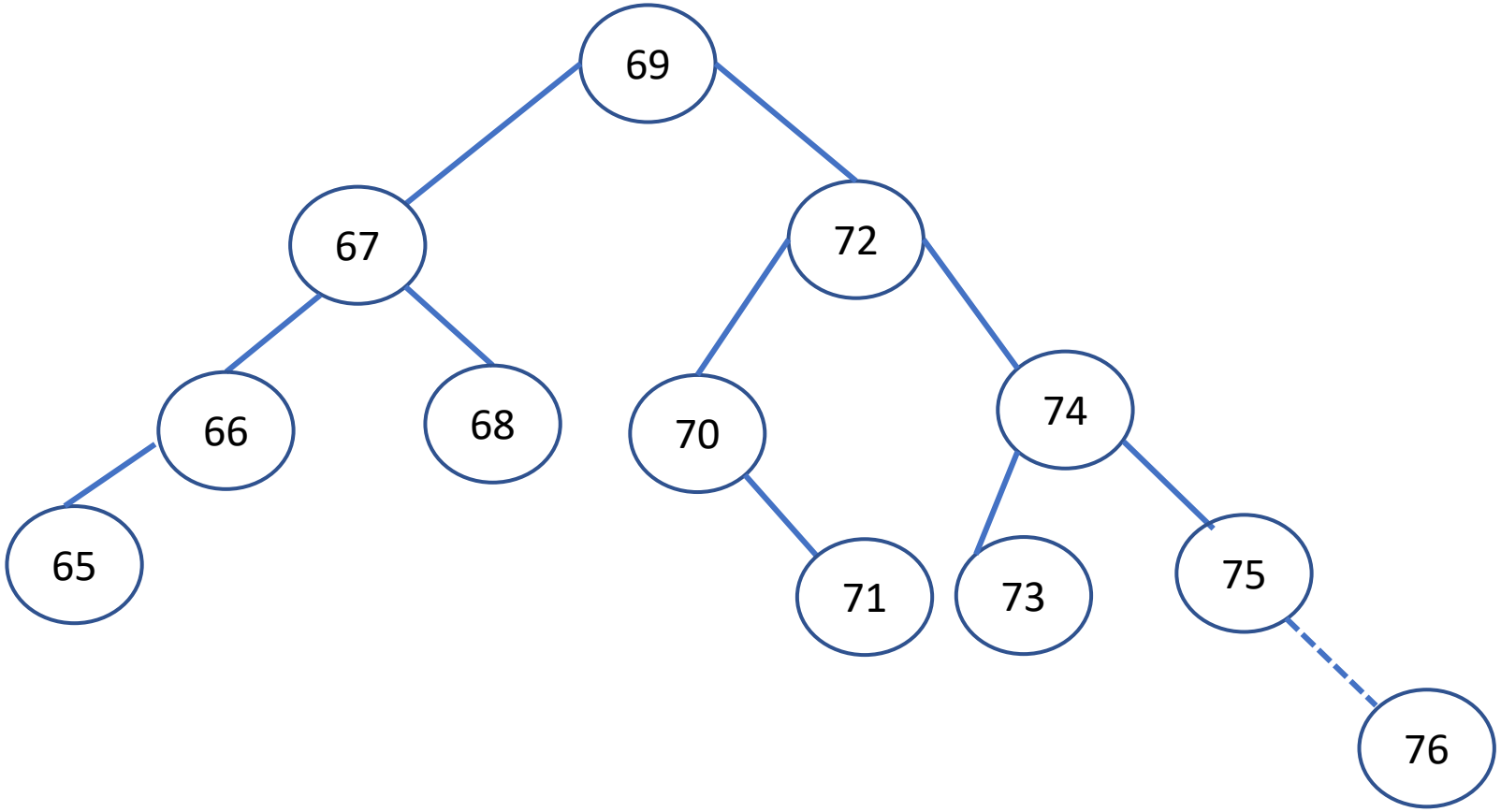


"***Left Rotation***"



AVL Tree: Construction

72, 73, 74, 66, 65, 69, 67, 70, 68, 71, 75, 76



Exercise: AVL Tree (1)

AVL tree is created by inserting the keys 13, 17, 12, 16, 14, 15, and 18 in the given order (Assume the tree is initially empty). Then the level order traversals of the tree would be.

A. *13, 12, 14, 16, 15, 17, 18*
B. *14, 13, 16, 12, 17, 15, 18*
C. *13, 12, 14, 15, 16, 17, 18*
D. *14, 13, 16, 12, 15, 17, 18*

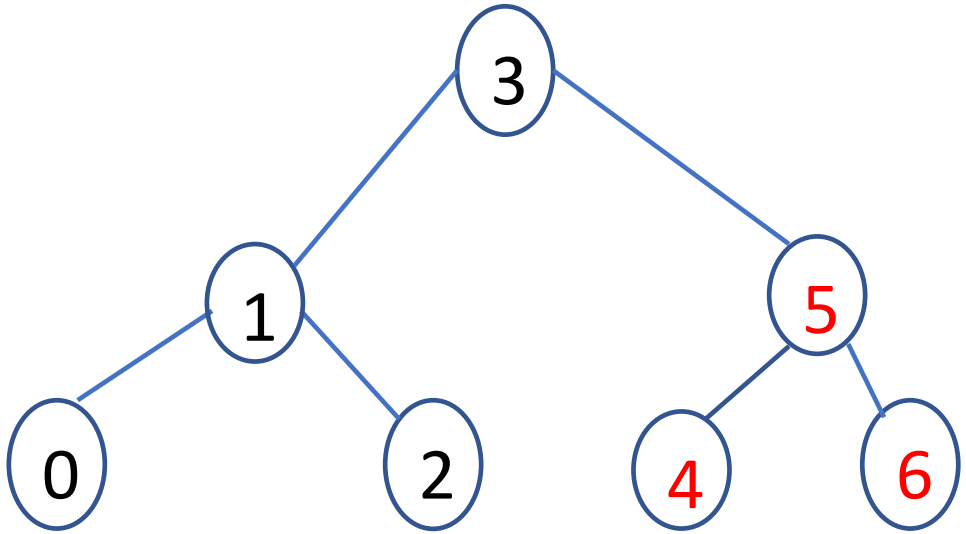
Exercise: AVL Tree (2)

AVL tree is created by inserting the keys 1, 7, 2, 6, 4, 5, and 8 in the given order (Assume the tree is initially empty). Then the level order traversals of the tree would be.

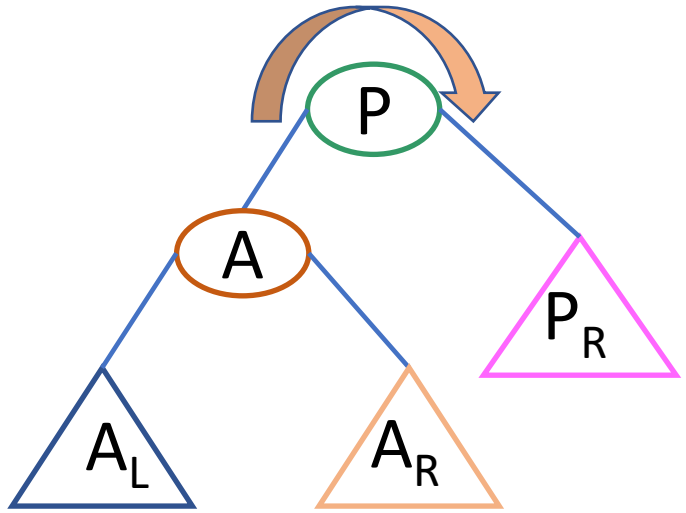
A. 4, 2, 6, 1, 5, 8, 7
B. 4, 2, 6, 1, 5, 7, 8
C. 4, 2, 1, 5, 6, 7, 8

Exercise: AVL Tree (3)

Consider the AVL tree given below, If we are deleting the keys 6,5,4 respectively, Minimum number of rotations is needed to make it a balanced AVL tree again ?.



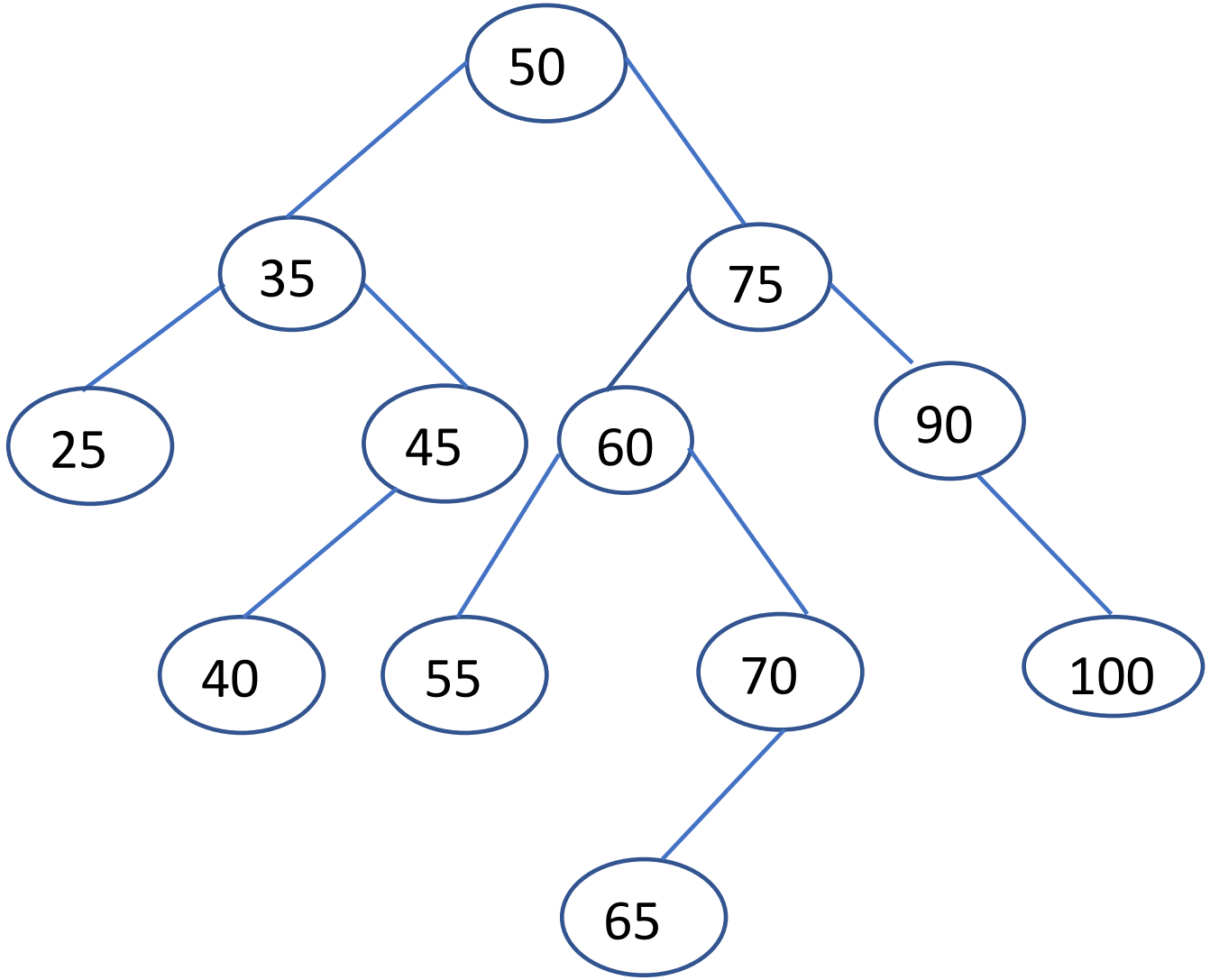
"***Right Rotation***"



A. 1
B. 2
C. 3
D. 0

Exercise: AVL Tree (4)

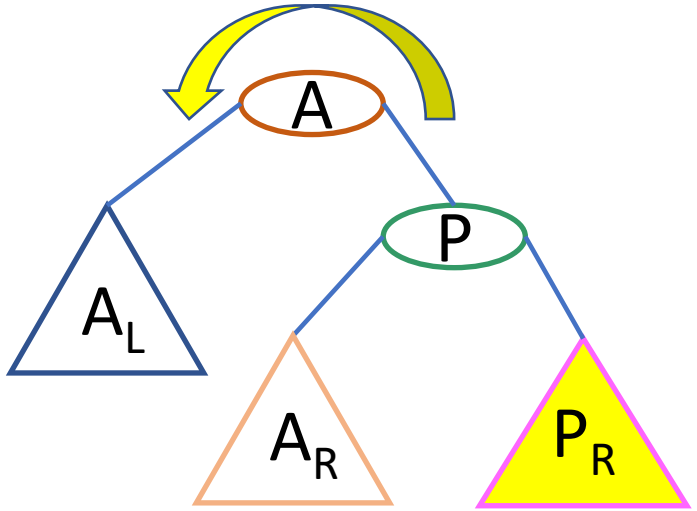
Consider the following AVL tree, after deleting node 45 what is the element at the root?.



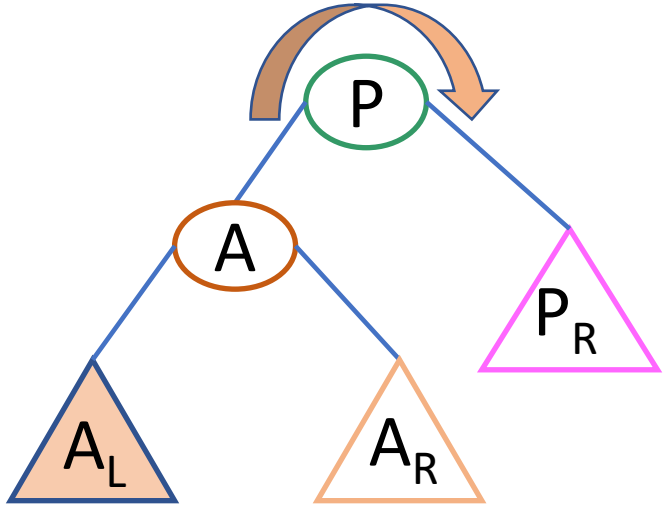
AVL Tree: Construction

8, 9, 10, 2, 1, 5, 6, 4, 7, 11, 12, 3

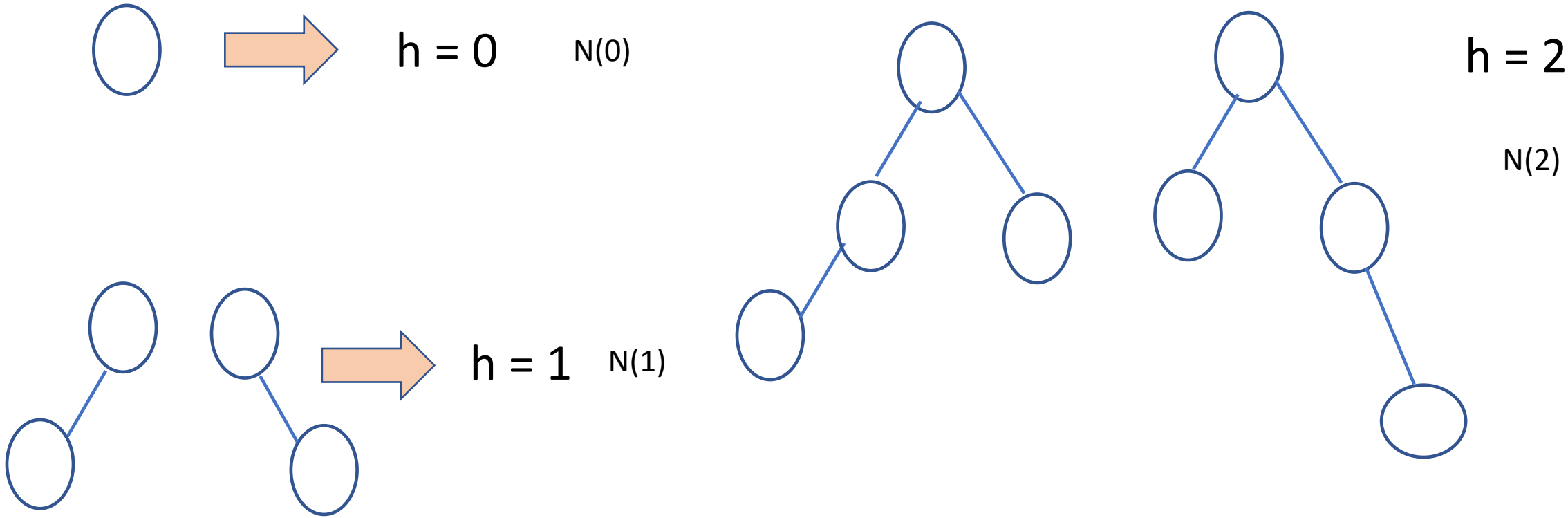
"***Left Rotation***"



"***Right Rotation***"

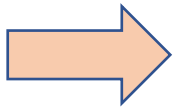


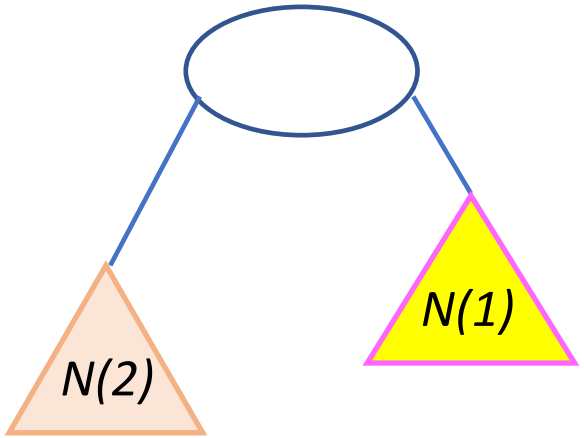
What is the minimum number of nodes in an AVL tree of height h?

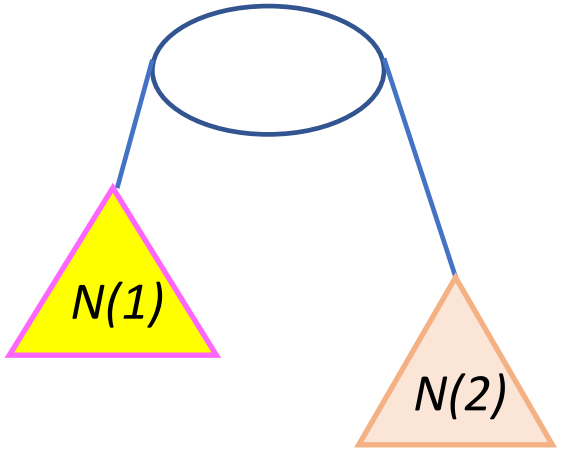


What is the minimum Number of nodes in an AVL tree of height h?

h = 3







$N(3) = N(2) + N(1) + 1$

$N(h) = N(h-1) + N(h-2) + 1$

What is the number of nodes in an AVL tree of height 6?

N(0) = 1
N(1) = 2
N(2) = 4
N(3) = 7
N(4) = 12

$$N(h) = N(h-1) + N(h-2) + 1$$

How many different trees can be formed with a minimal AVL tree height h?

N(0) = 1
N(1) = 2

N (2) = 2 N(1) * N (0)*

N (h) = 2 N(h-1) * N (h-2); h $\geq$ 2*

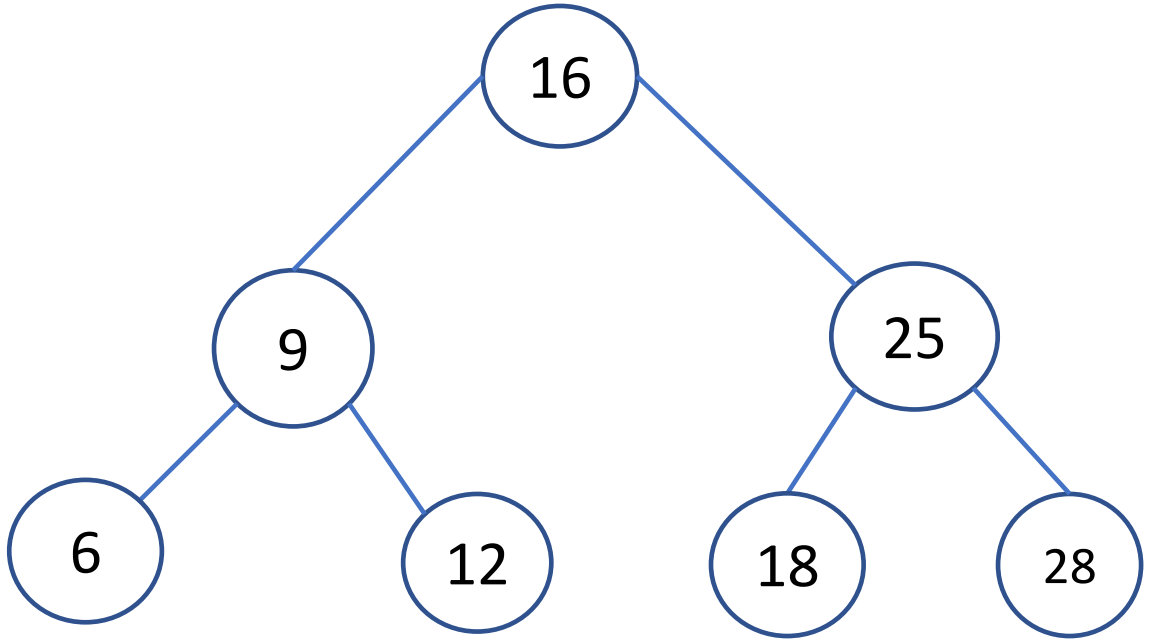
Consider the following statements

S1 : An insertion in an AVL with n nodes requires Θ(n) rotations.
S2 : Finding the minimum value in an AVL tree containing n elements takes O(logn) time
S3 : Both Insert and find in an AVL tree takes O(logn) time.
S4: Finding the k-th largest item in a standard AVL tree implementation containing n elements takes O(n) time.
S5 : A set of numbers are inserted into an empty BST in sorted order and inserted into an empty AVL tree in random order. Listing all elements in sorted order from the BST is O (n), while listing them in sorted order from the AVL tree is O(logn)

Example: Binary Search Tree

Successor (S, x) : Given an element x whose key is from a totally ordered set S, returns a pointer to the next larger element in S, or NULL if x is the maximum element.

Predecessor (S, x) : Given an element x whose key is from a totally ordered set S, returns a pointer to the next smaller element in S, or NULL if x is the minimum element.



thank you!

email:
k.kondepu@iitdh.ac.in