AVL Tree

Hyoungshick Kim
College of Software
Sungkyunkwan University

Complexity of Dictionary Operations
Get(), Insert() and Delete()

Data Structure	Worst Case	Expected
Hash Table	O(n)	O(1)
Binary Search Tree	O(n)	O(log n)
Balanced Binary Search Tree	O(log n)	O(log n)

n is number of elements in dictionary

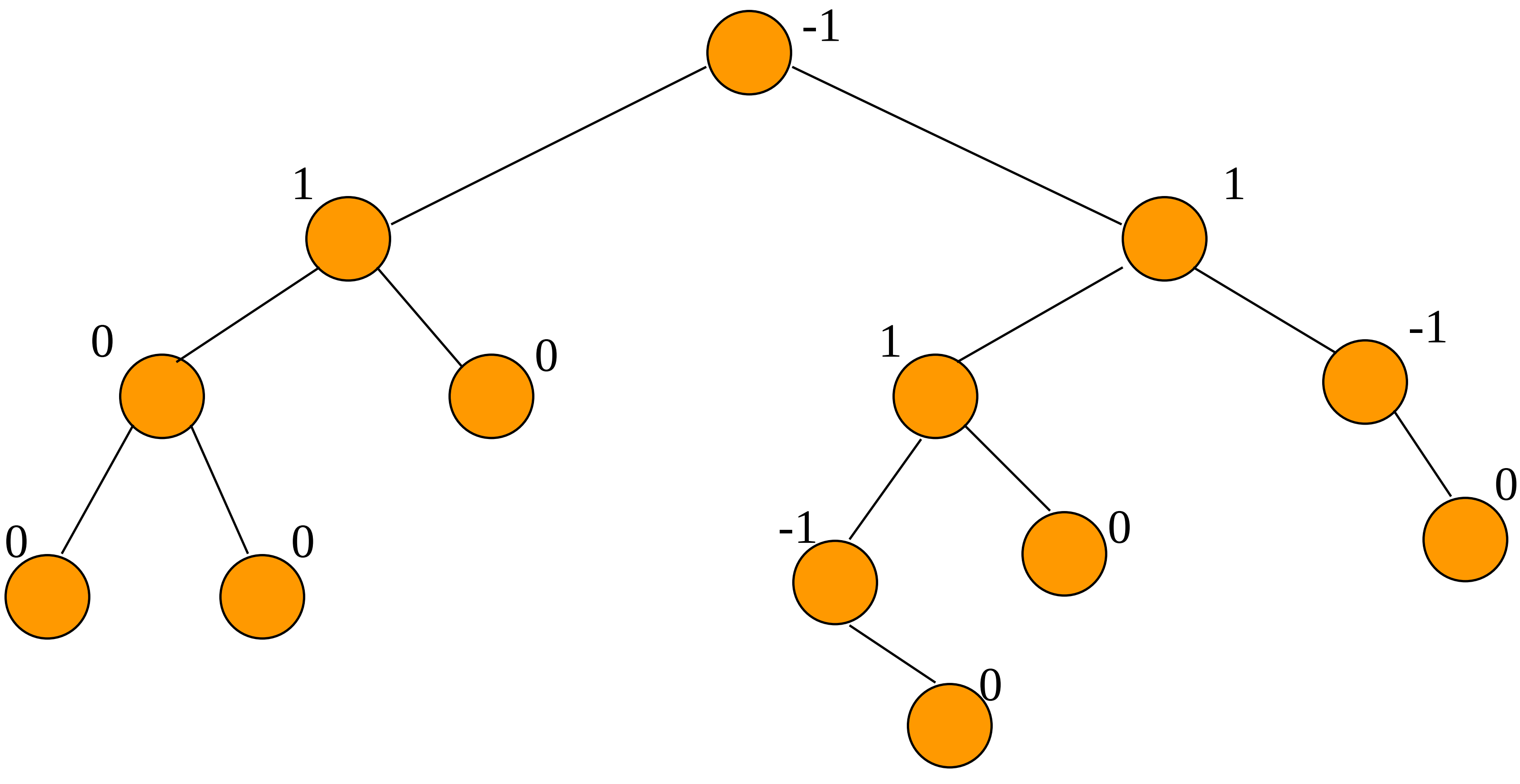
AVL Tree

- (self-balancing) binary tree
- Georgy Adelson-Velsky and Landis' tree, named after the inventors
- for every node x, define its balance factor

  balance factor of x = height of left subtree of x

  – height of right subtree of x
- balance factor of every node x is – 1, 0, or 1

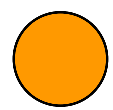
Balance Factors



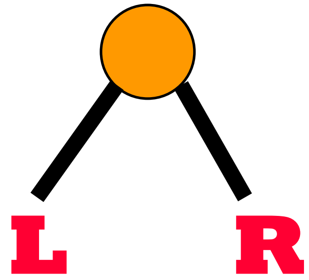
This is an AVL tree.

Proof of Upper Bound on Height

- Let N_h = min # of nodes in an AVL tree whose height is h.
- $N_0 = 0$.
- $N_1 = 1$.



N_h, $h > 1$

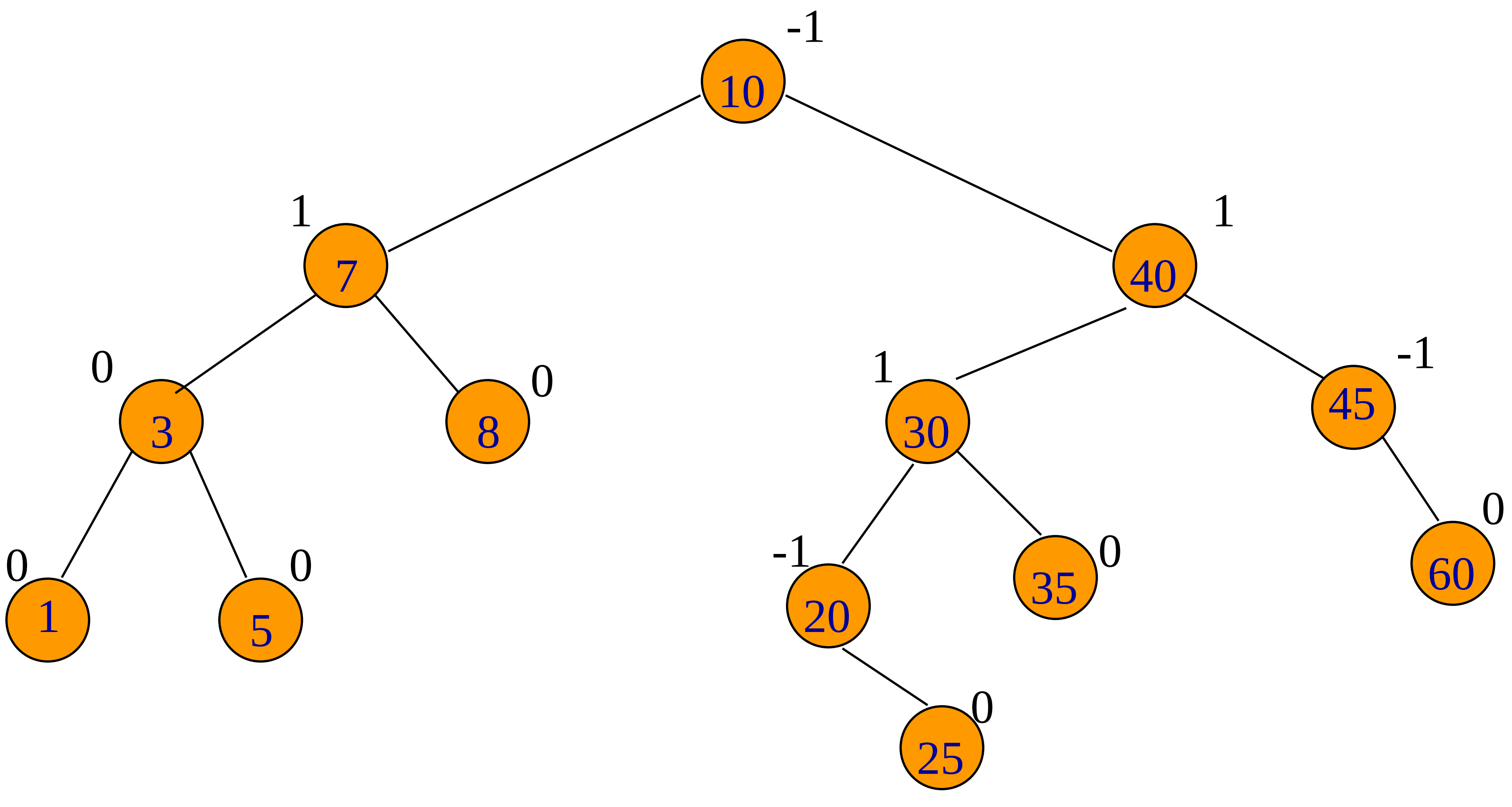


- Both L and R are AVL trees.
- The height of one is h-1.
- The height of the other is h-2.
- The subtree whose height is h-1 has N_{h-1} nodes.
- The subtree whose height is h-2 has N_{h-2} nodes.
- So, $N_h = N_{h-1} + N_{h-2} + 1$.

- $N_h = N_{h-1} + N_{h-2} + 1$
- $N_{h-1} = N_{h-2} + N_{h-3} + 1$
- $N_h = (N_{h-2} + N_{h-3} + 1) + N_{h-2} + 1$
- $N_h > 2 \cdot N_{h-2}$
- $N_h > 2 \cdot 2 \cdot N_{h-4}$
- $N_h > 2^{h/2}$
- $\log N_h > \log 2^{h/2}$
- $2 \cdot \log N_h > h$
- $h = O(\log N_h)$

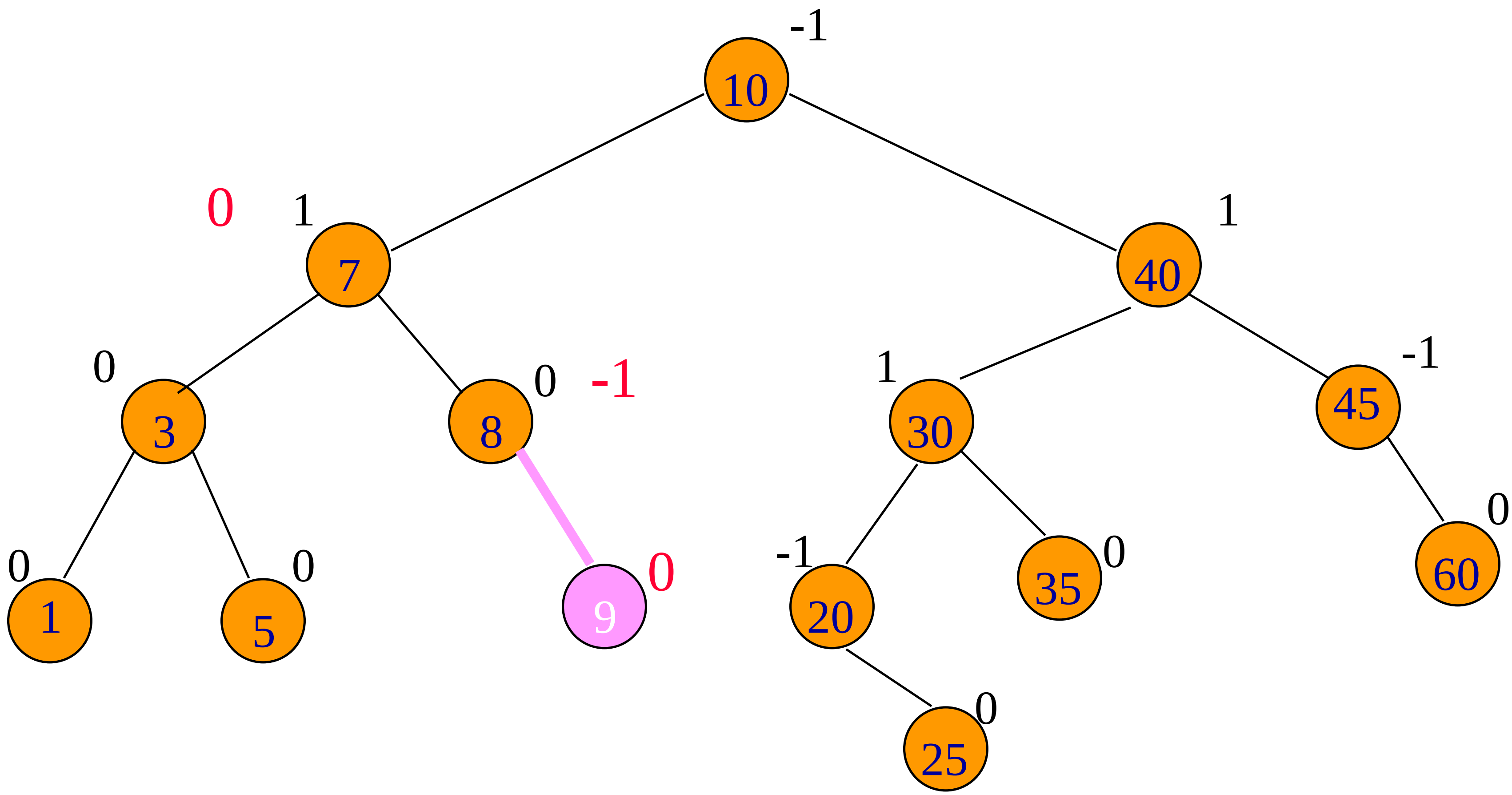
The height of an AVL tree with n nodes is $O(\log_2 n)$.

AVL Search Tree



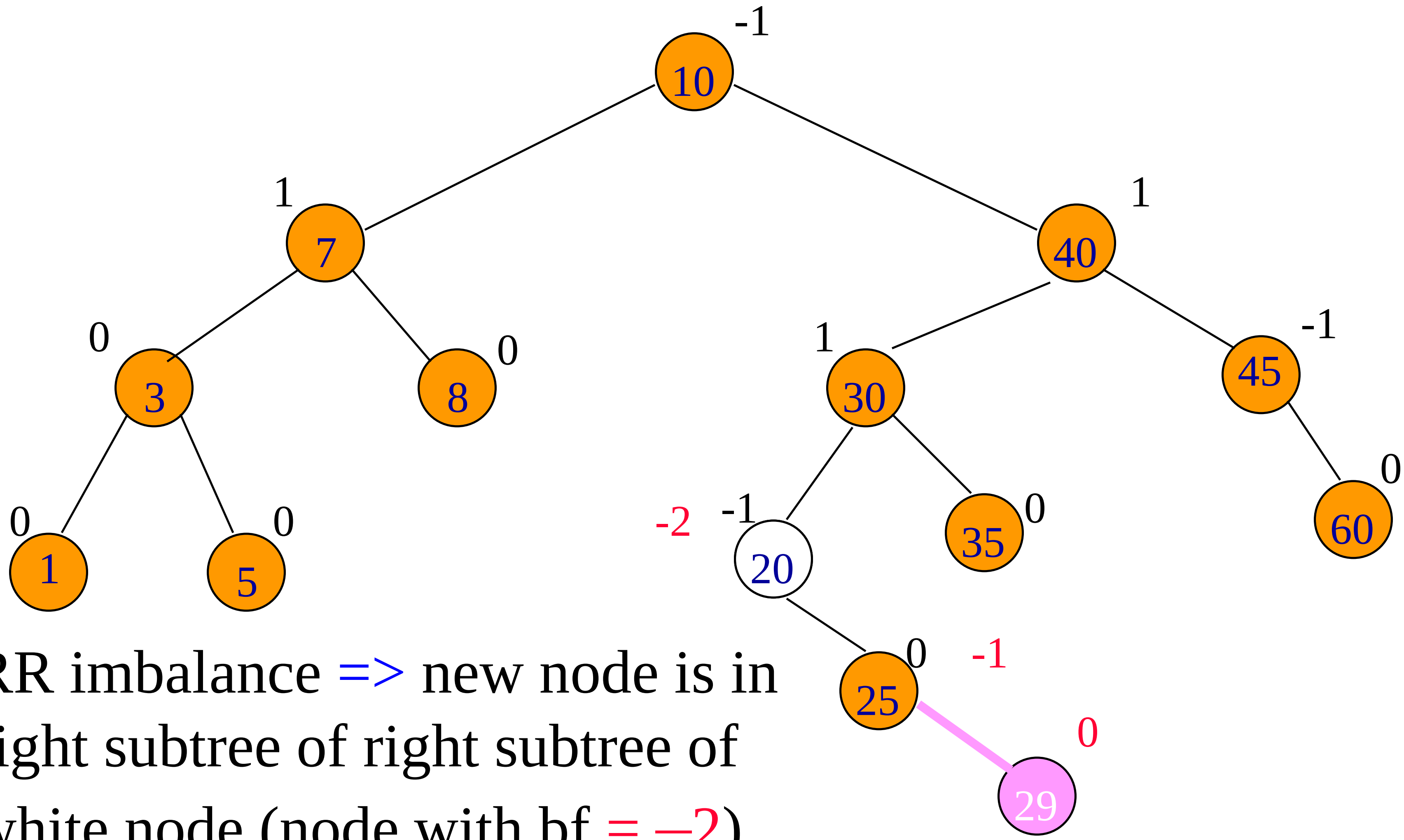
Search operation can be performed in the same manner.

Insert(9)



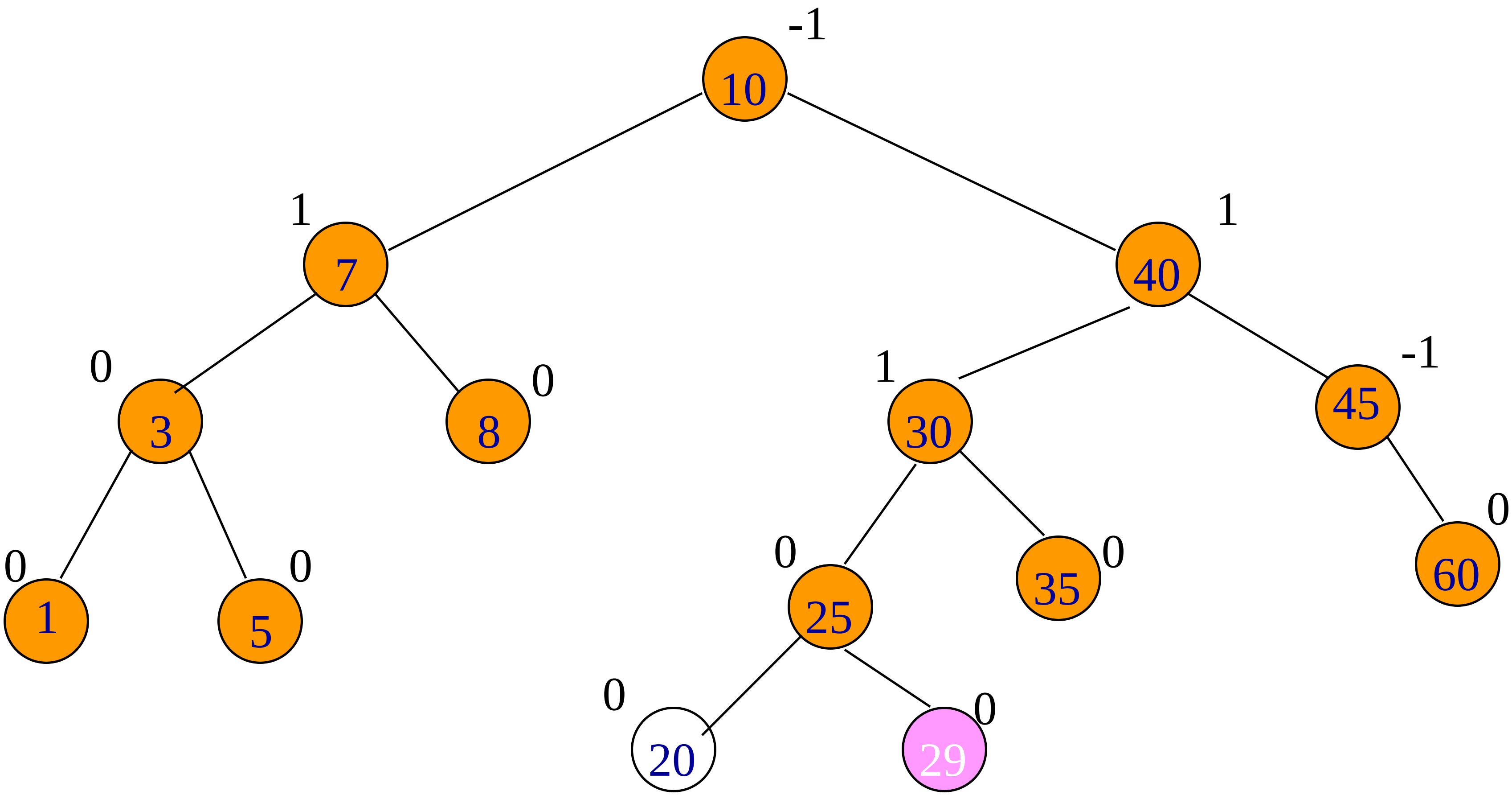
We need to go further up the tree
to adjust balance factors.

Insert(29)



RR imbalance => new node is in right subtree of right subtree of white node (node with bf = –2)

Insert(29)



RR rotation.

Insert

- Following insert, retrace path towards root and adjust balance factors as needed.
- Stop when you reach a node whose balance factor becomes 0, 2, or –2, or when you reach the root.
- The new tree is not an AVL tree only if you reach a node whose balance factor is either 2 or –2.
- In this case, we say the tree has become unbalanced.

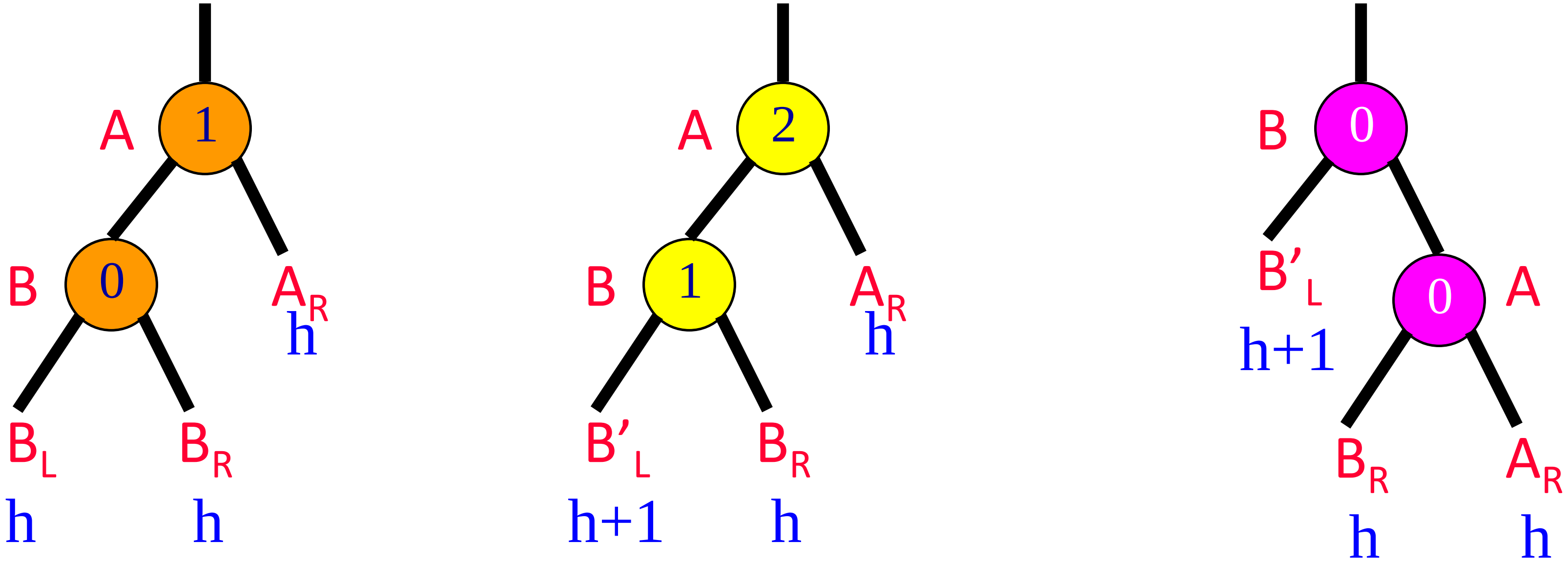
A-Node

- Let A be the nearest ancestor of the newly inserted node whose balance factor becomes +2 or –2 following the insert.

Imbalance Types

- RR … newly inserted node is in the right subtree of the right subtree of A.
- LL … left subtree of left subtree of A.
- RL… left subtree of right subtree of A.
- LR… right subtree of left subtree of A.

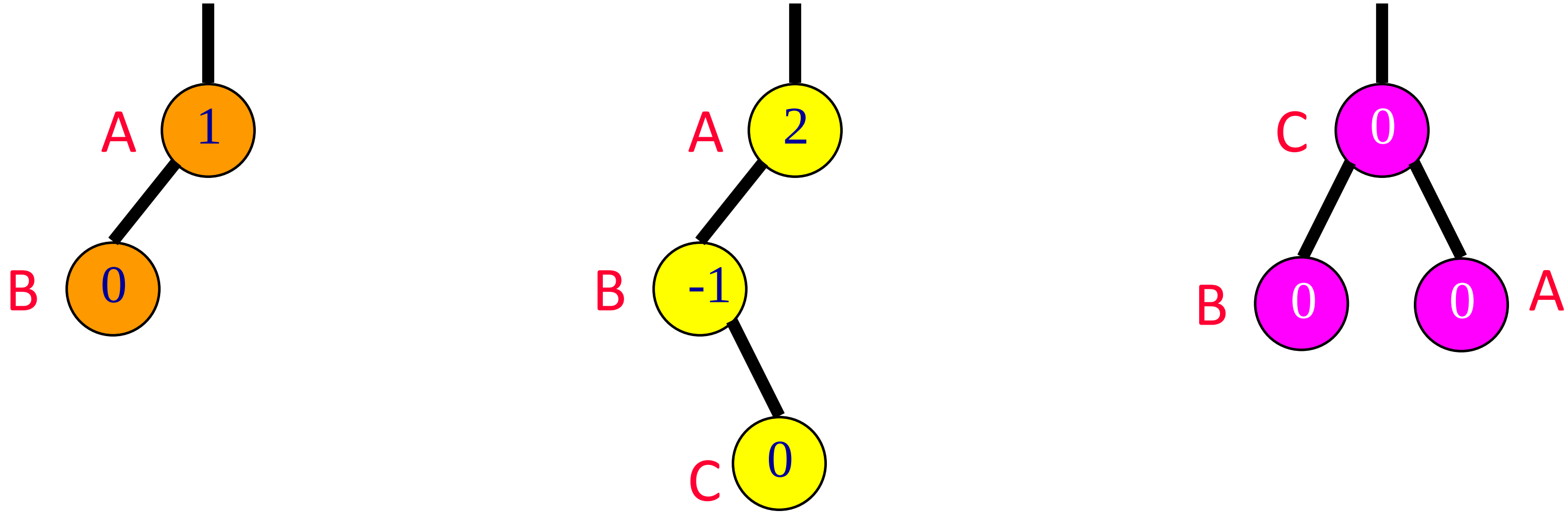
LL Rotation



Before insertion. After insertion. After rotation.

- Subtree height is unchanged.
- No further adjustments to be done.

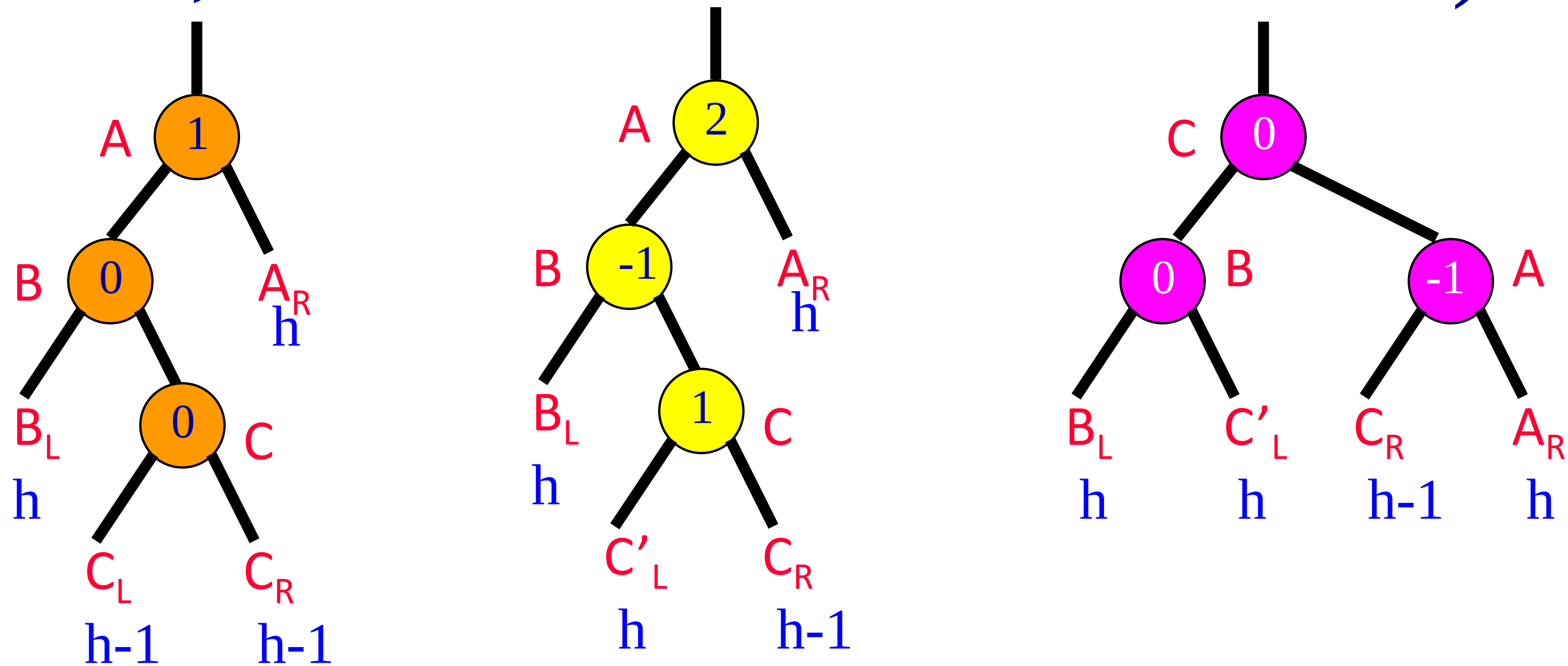
LR Rotation (case 1: B is a leaf)



Before insertion. After insertion. After rotation.

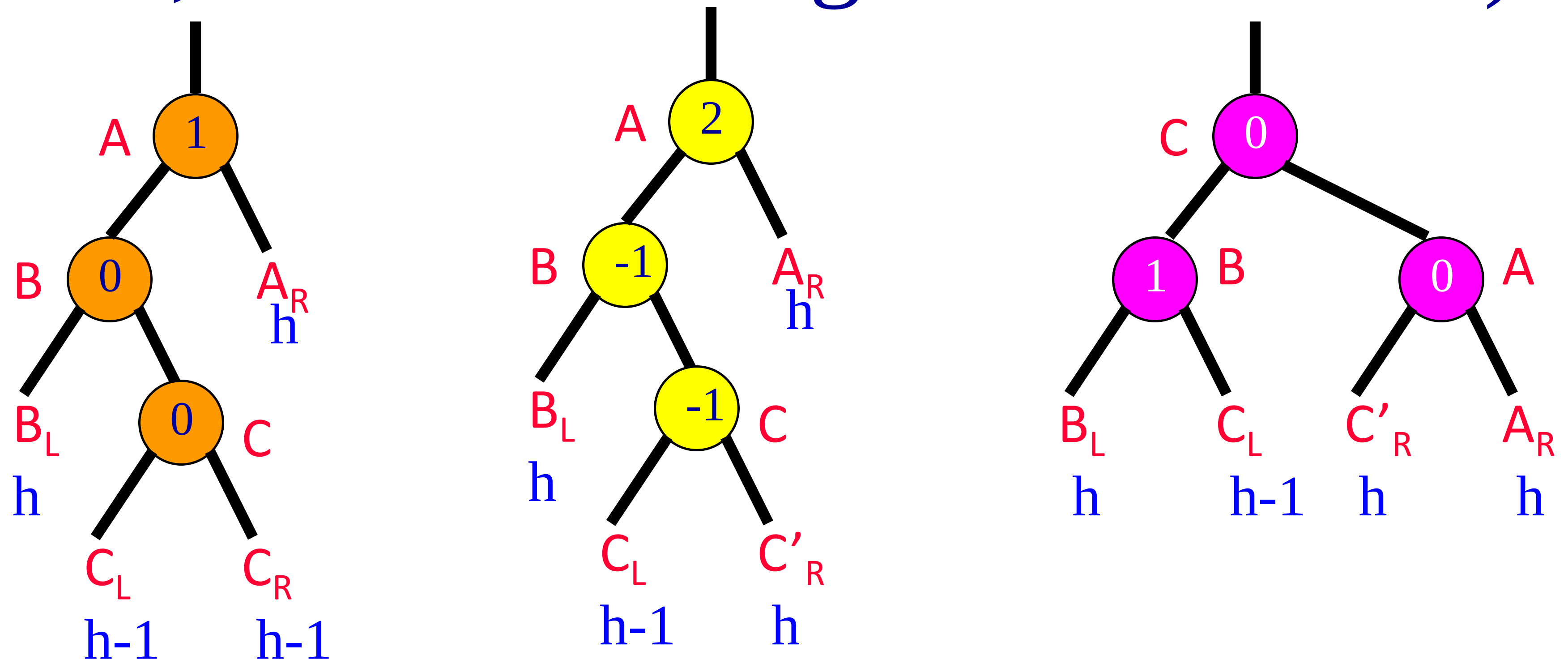
- Subtree height is unchanged.
- No further adjustments to be done.

LR Rotation (case 2: B is not a leaf, insert in the left subtree of C)



- Subtree height is unchanged.
- No further adjustments to be done.

LR Rotation (case 3: B is not a leaf, insert in the right subtree of C)

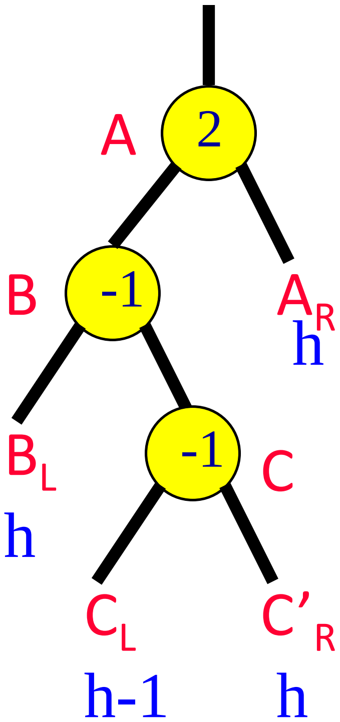


- Subtree height is unchanged.
- No further adjustments to be done.

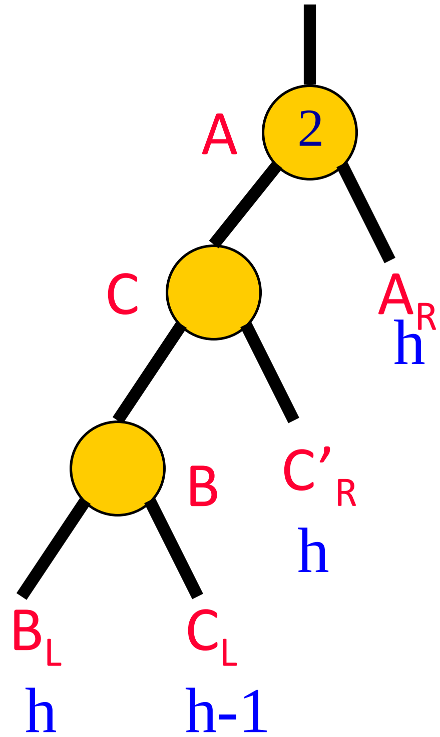
Single & Double Rotations

- Single
  - LL and RR
- Double
  - LR and RL
  - LR is RR followed by LL
  - RL is LL followed by RR

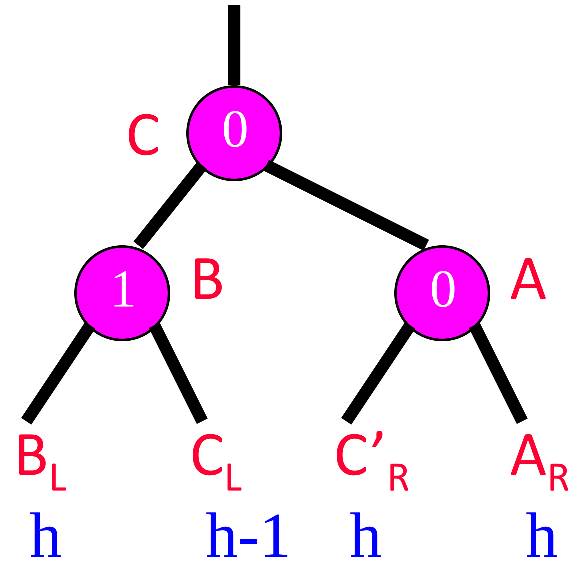
LR is RR + LL



After insertion.



After RR rotation.

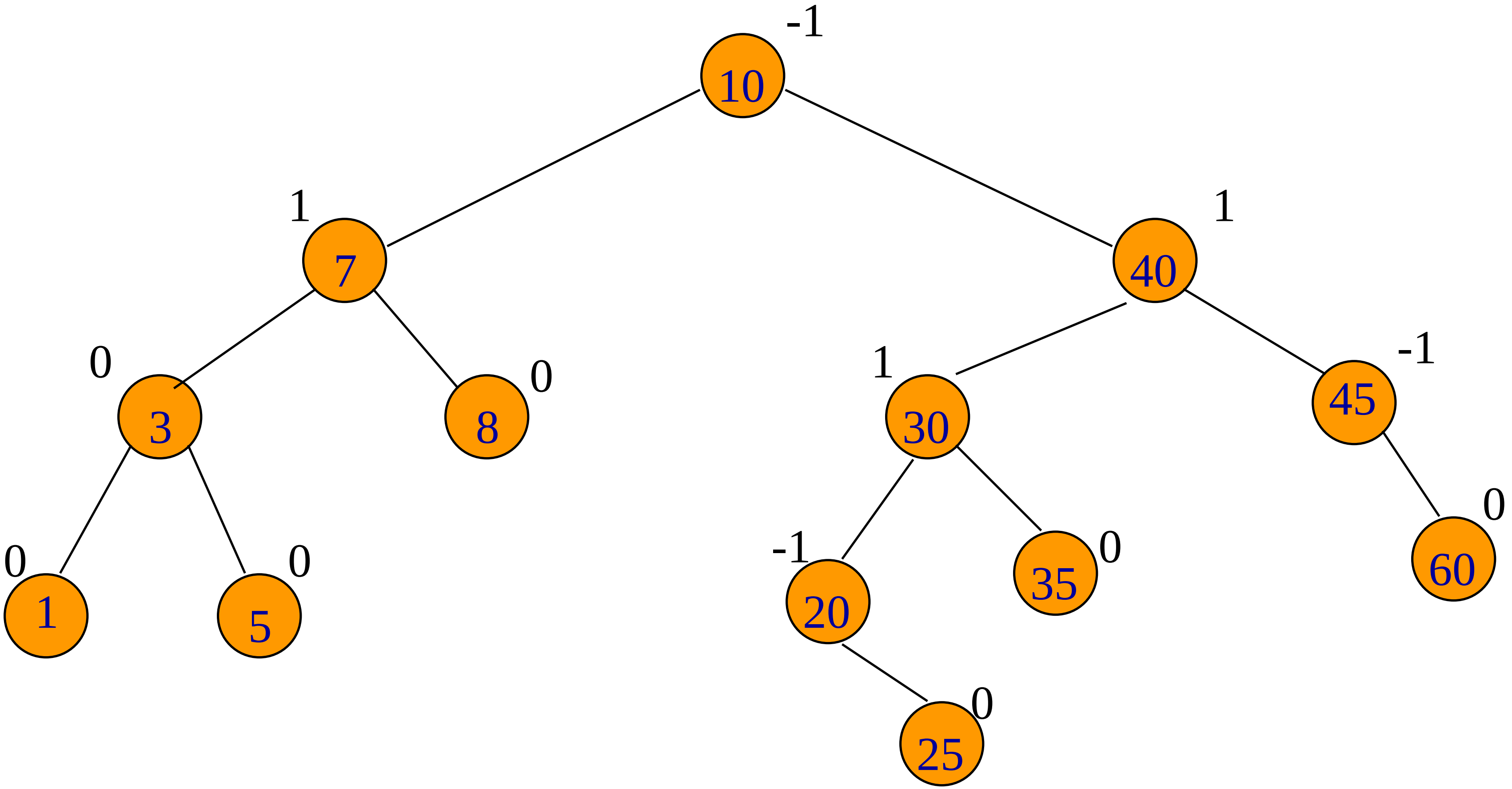


After LL rotation.

Rotation Frequency

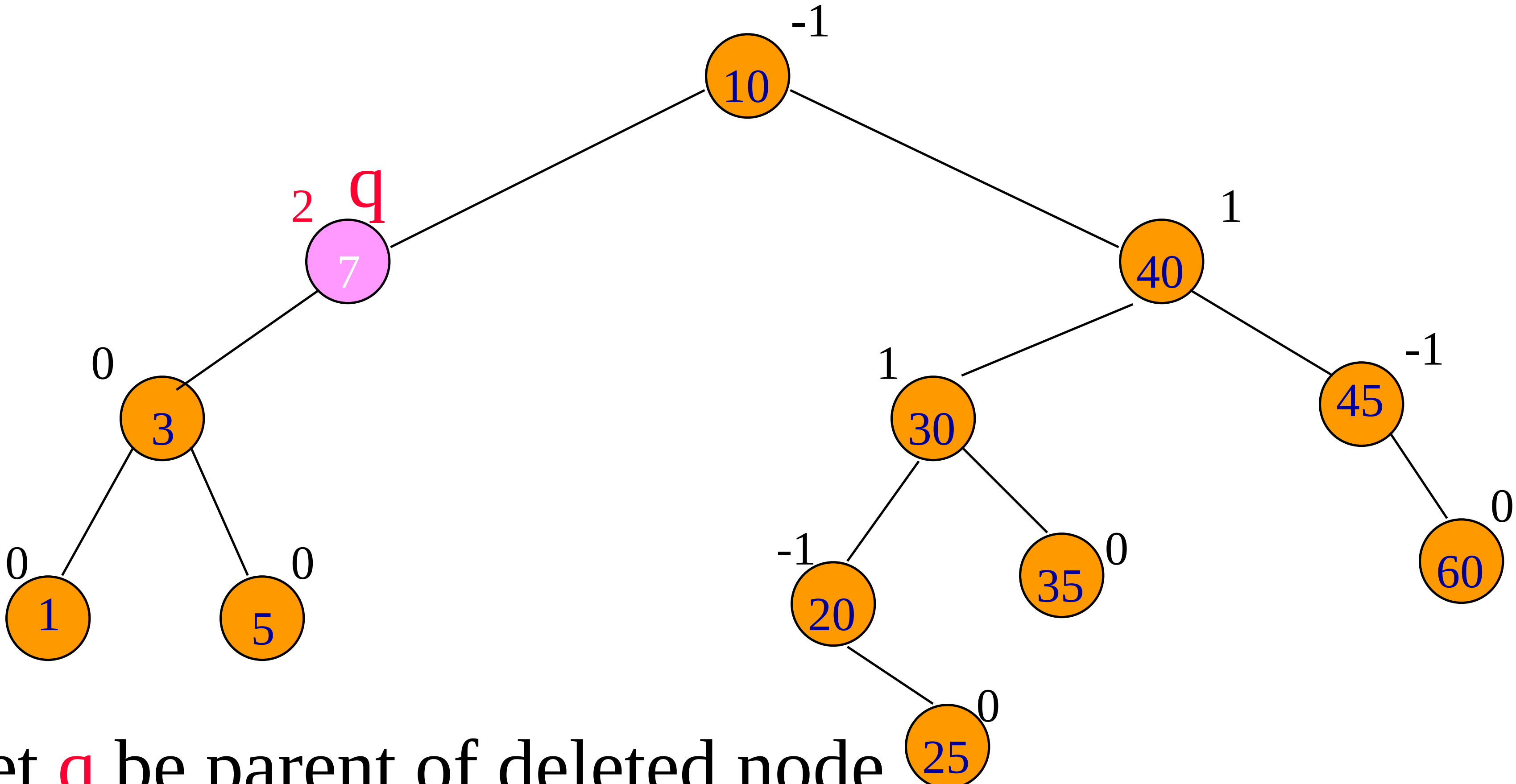
- Insert random numbers.
  - No rotation … 53.4% (approx).
  - LL/RR … 23.3% (approx).
  - LR/RL … 23.2% (approx).

Delete an Element



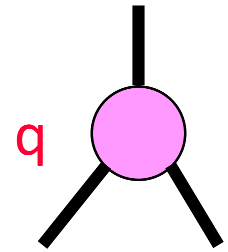
Delete 8.

Delete an Element



- Let q be parent of deleted node.
- Retrace path from q towards root.

New Balance Factor of q

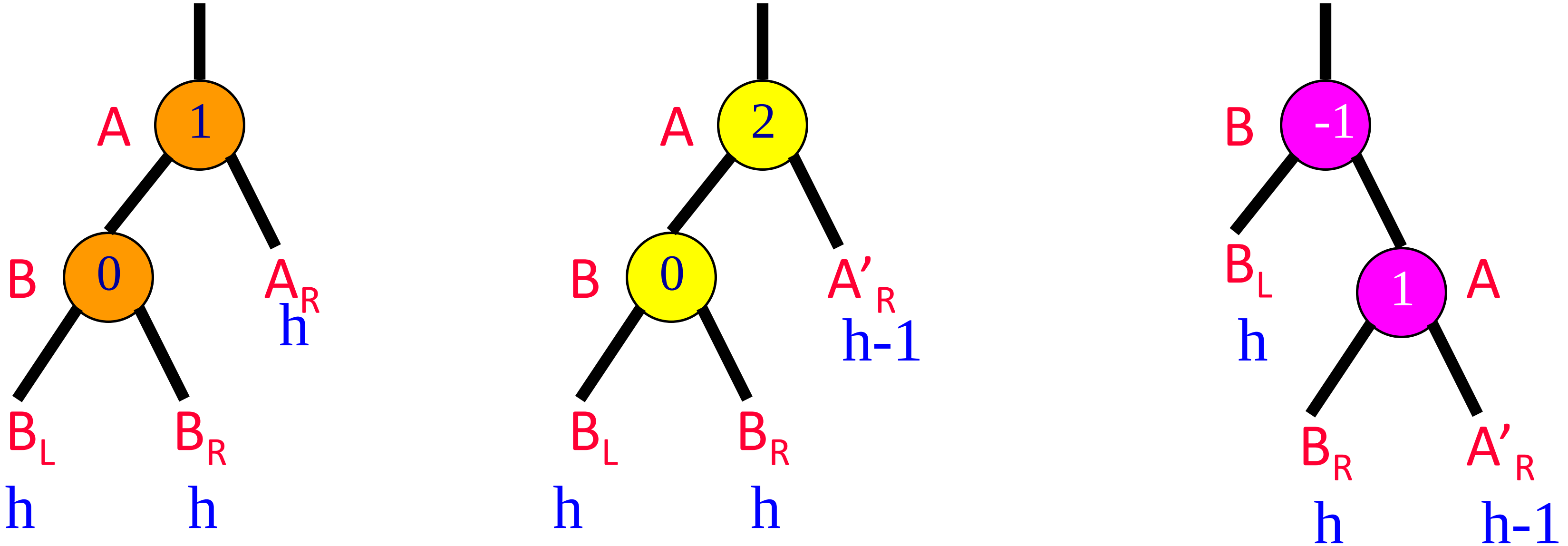


- Deletion from left subtree of q => bf--.
- Deletion from right subtree of q => bf++.
- New balance factor = 1 or –1 => no change in height of subtree rooted at q.
- New balance factor = 0 => height of subtree rooted at q has decreased by 1.
- New balance factor = 2 or –2 => tree is unbalanced at q.

Imbalance Classification

- Let A be the nearest ancestor of the deleted node whose balance factor has become 2 or –2 following a deletion.
- Deletion from left subtree of A => type L.
- Deletion from right subtree of A => type R.
- Type R => new bf(A) = 2.
- So, old bf(A) = 1.
- So, A has a left child B.
  - bf(B) = 0 => R0.
  - bf(B) = 1 => R1.
  - bf(B) = –1 => R-1.

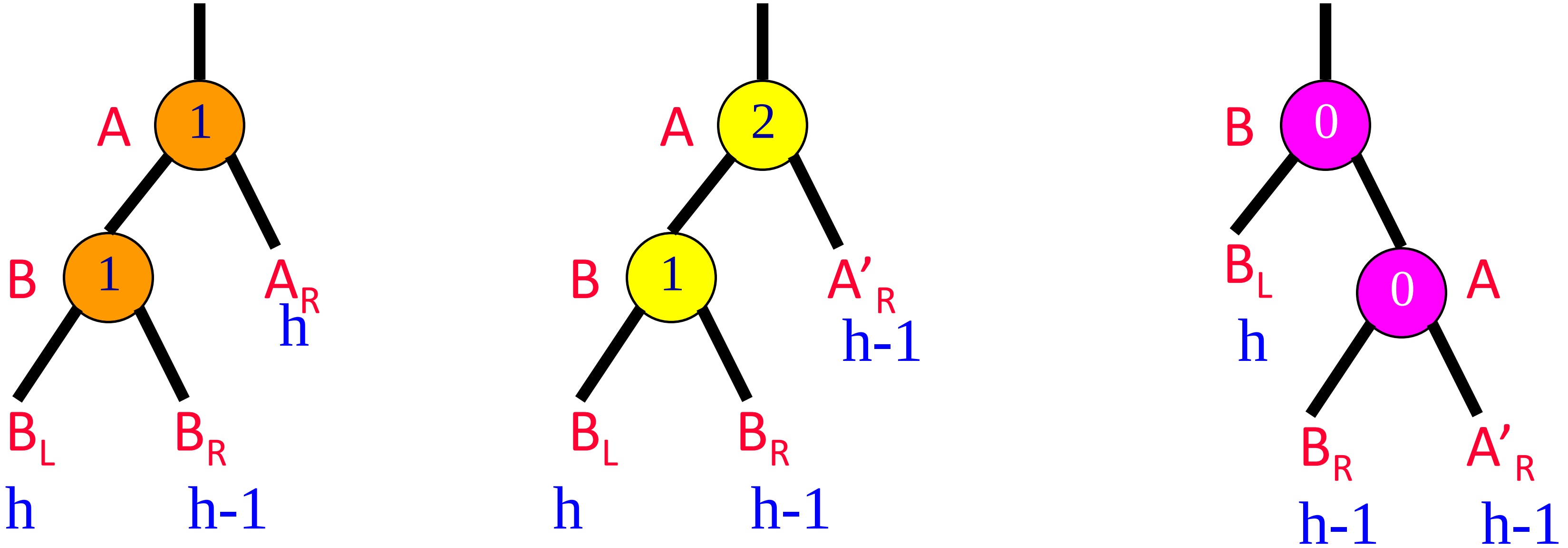
R0 Rotation



Before deletion. After deletion. After rotation.

- Subtree height is unchanged.
- No further adjustments to be done.
- Similar to LL rotation.

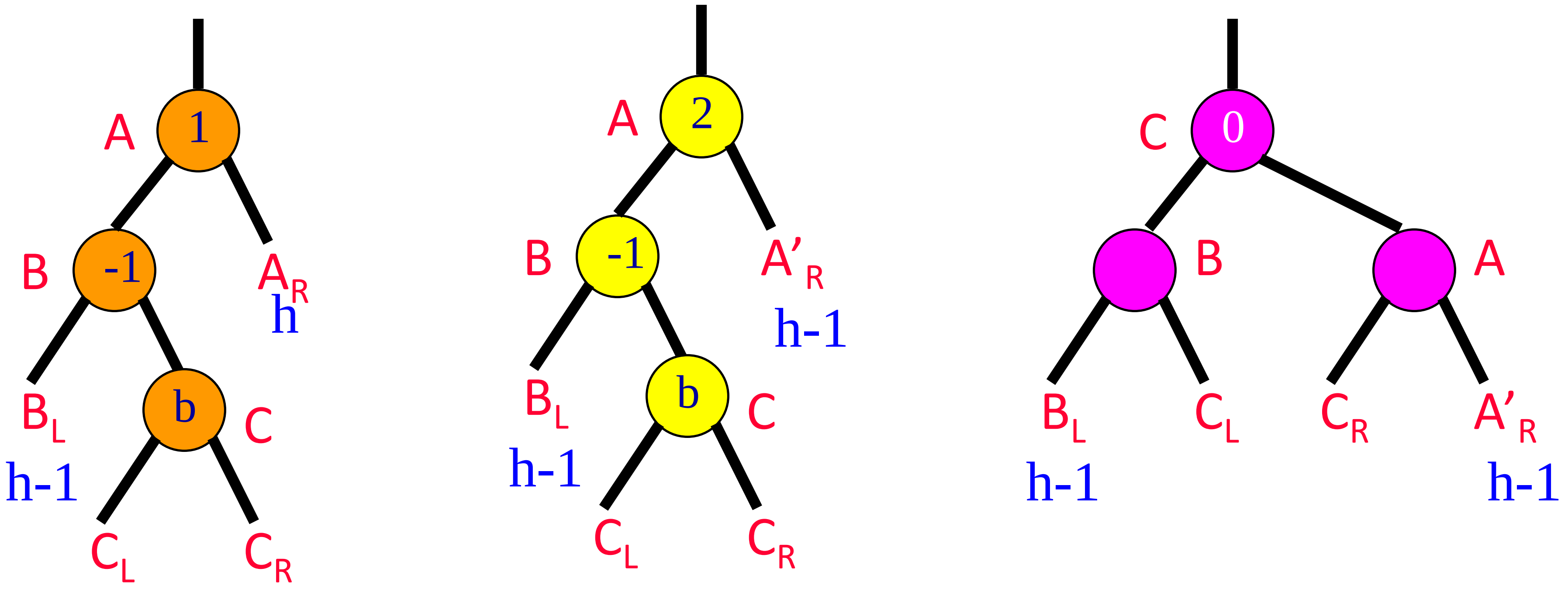
R1 Rotation



Before deletion. After deletion. After rotation.

- Subtree height is reduced by 1.
- Must continue on path to root.
- Similar to LL and R0 rotations.

R-1 Rotation



- New balance factor of A and B depends on b.
- Subtree height is reduced by 1.
- Must continue on path to root.
- Similar to LR.