Tree

- *introduction*
- *binary tree*
- *complete binary tree*
  - *max heap, min heap*
  - *Chapter 7 – heap sorting*
  - *Chapter 9 - priority queues*
- *binary search tree(bst)*
- **AVL tree -** *Chapter 10 – Efficient BST*

BST

- **Definition: A binary search tree is a binary tree in symmetric order.**

  - A binary tree is either
    - empty
    - a key-value pair and two binary trees [neither of which contain that key]

  equal keys ruled out

  node

  - Symmetric order means that
    - every node has a key
    - every node's key is
      larger than all keys in its left subtree
      smaller than all keys in its right subtree
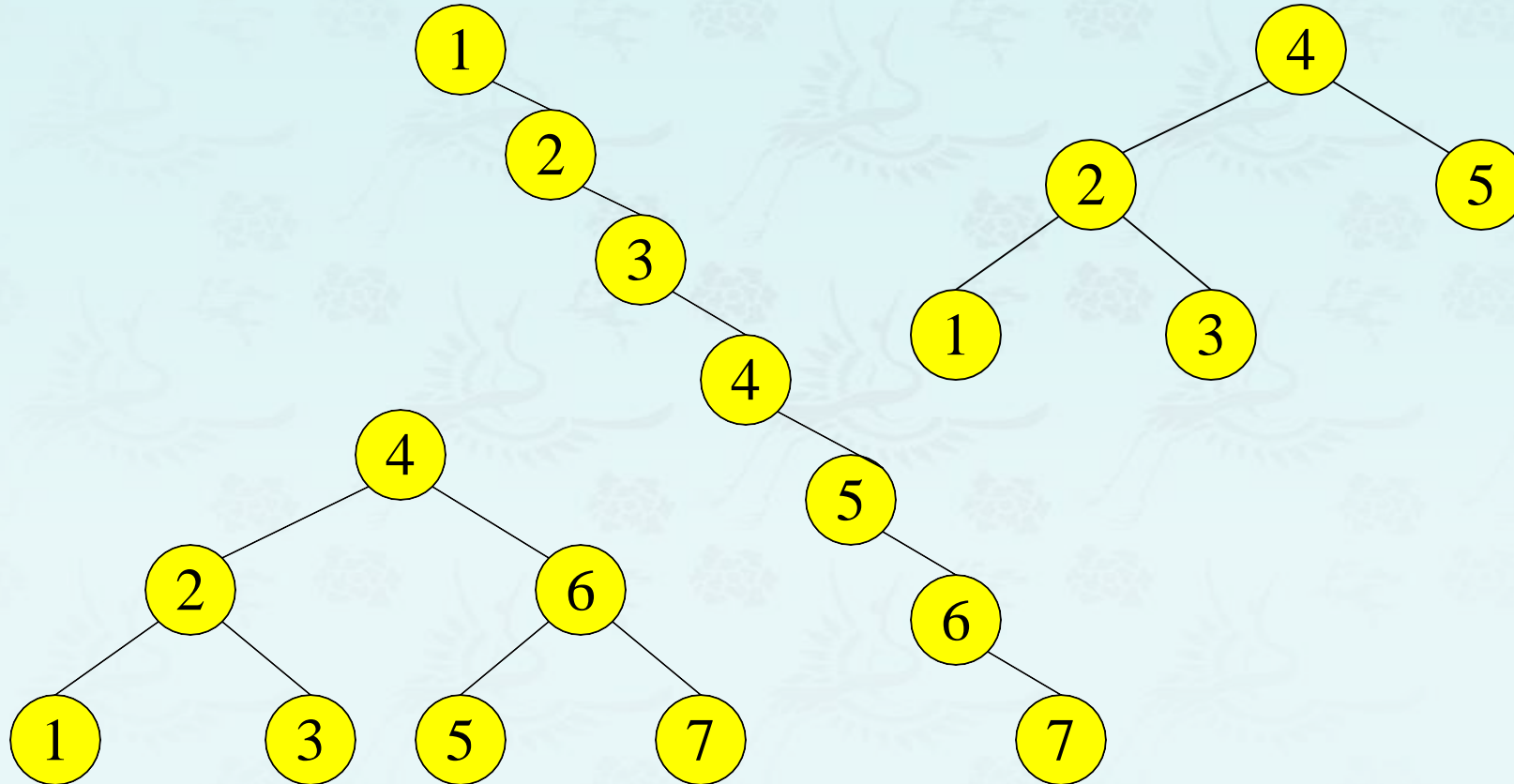
  subtrees

  smaller   larger

BST
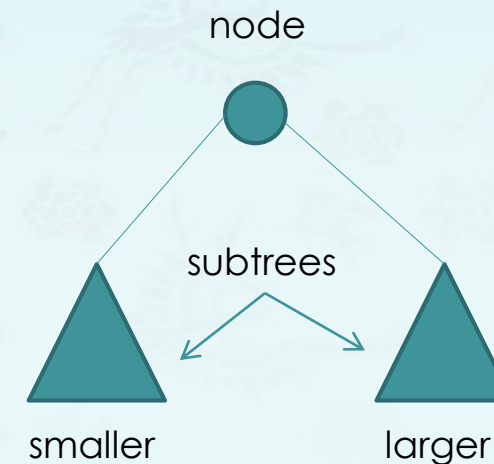
- **Definition: A binary search tree is a binary tree in symmetric order.**

- All BST operations are O(d), where d is tree depth

- Minimum d is $d = \lfloor \log_2 N \rfloor$ for a binary tree with N nodes

  - What is the best case tree?
  - What is the worst case tree?

- So, best case running time of BST operations is O(log N)

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

7

BST

Worst case running time is O(N)

- What happens when you Insert elements in ascending order?
  - Insert: 2, 4, 6, 8, 10, 12 into an empty BST
- Problem: Lack of "balance";
  - compare depths of left and right subtree
- Unbalanced degenerate tree

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

8

Balanced and unbalanced BST

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

9

Approaches to balancing trees

- ### Don't balance
  - May end up with some nodes very deep

- ### Strict balance
  - The tree must always be balanced perfectly

- ### Pretty good balance
  - Only allow a little out of balance

- ### Adjust on access
 - Self-adjusting

Balancing Binary Search Trees

Many algorithms exist for keeping BST balanced

- Adelson-Velskii and Landis (**AVL**) trees (height-balanced trees)
- Weight-balanced trees
- **Red-black** trees;
- **Splay** trees and other self-adjusting trees
- **B-trees** and other (e.g. 2-4 trees) multiway search trees

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

11

Perfect Balance

- Want a complete tree after every operation
  - tree is full except possibly in the lower right
- This is expensive
  - For example, insert 2 in the tree on the left and then rebuild as a complete tree

Insert 2 & complete tree
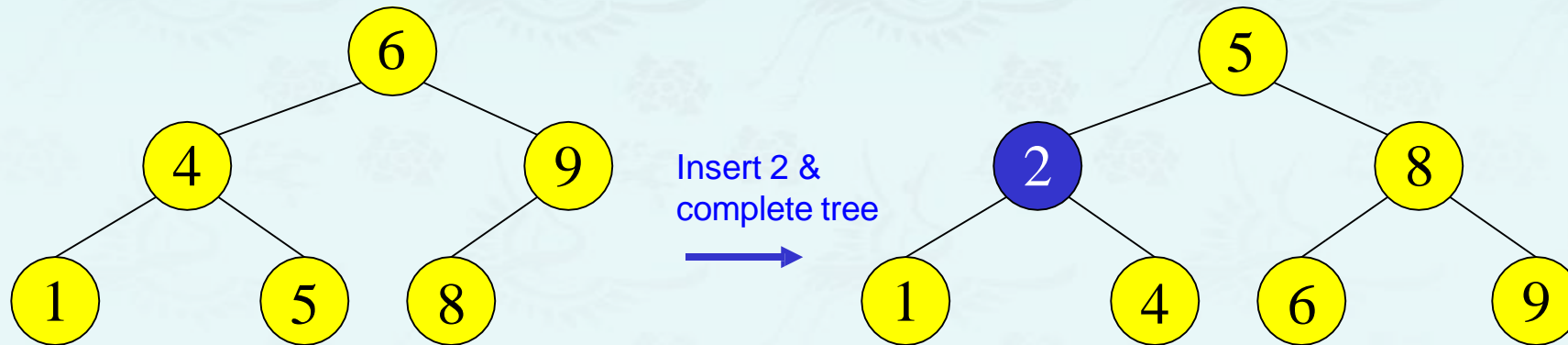
AVL Trees (1962)

- Named after 2 Russian mathematicians
- Georgii **A**delson-**V**elsky (1922 - 2014)
- Evgenii Mikhailovich **L**andis (1921-1997)

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

14

AVL - Good but not Perfect Balance

- Height-balanced binary search trees
- Balance factor of a node
  - height(left subtree) - height(right subtree)
- For every node, heights of left and right subtree can differ by no more than 1
  - Store current heights in each node or
    compute it on the fly

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

15

Node Heights
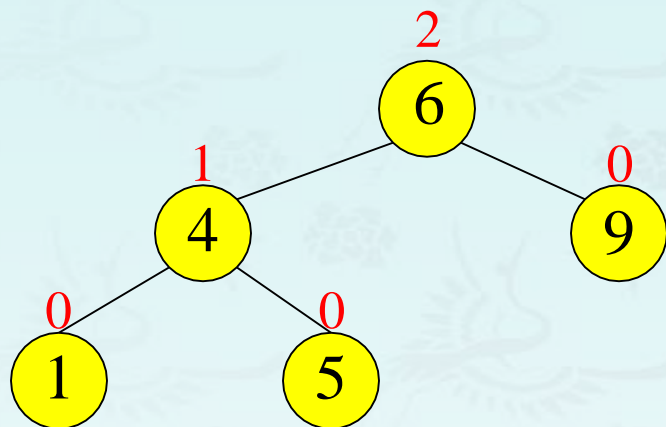
Tree A (AVL)

Tree B (AVL)

height of node = h
balance factor = h_{left} - h_{right}
empty height = -1

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

16

Node Heights after Insert 7

Tree A (AVL)   Tree B (AVL)
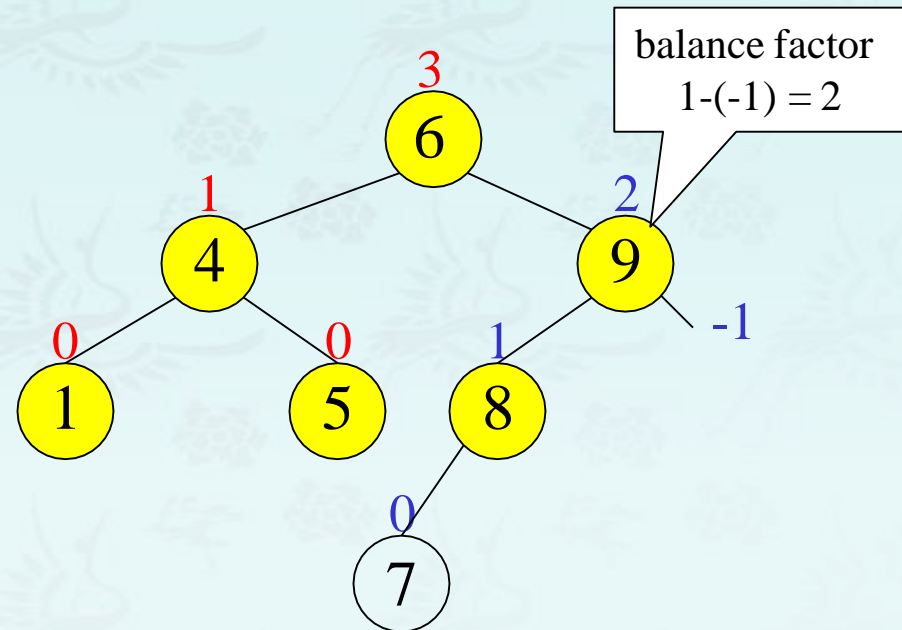
balance factor
$1-(-1) = 2$

height of node = h
balance factor = h_{left} - h_{right}
empty height = -1

Insert and Rotation in AVL Trees

- Insert operation may cause balance factor to become 2 or −2 for some node
  - Only nodes on the path from insertion point to root node have possibly changed in height
  - So after the Insert, **go back up** to the root node by node.
  - If a new balance factor (the difference h_{left} - h_{right}) is 2 or −2, adjust tree by **rotation** around the node

Single Rotation in an AVL Tree

balance factor
0-0 = 0

LL
Single Rotation
Right Rotation

Single Rotation in an AVL Tree

Left Left Case

5 1 → 2

L D

L 4 1(0)

C

3

A B

Balanced

4 0(-1)

3 5

A B C D
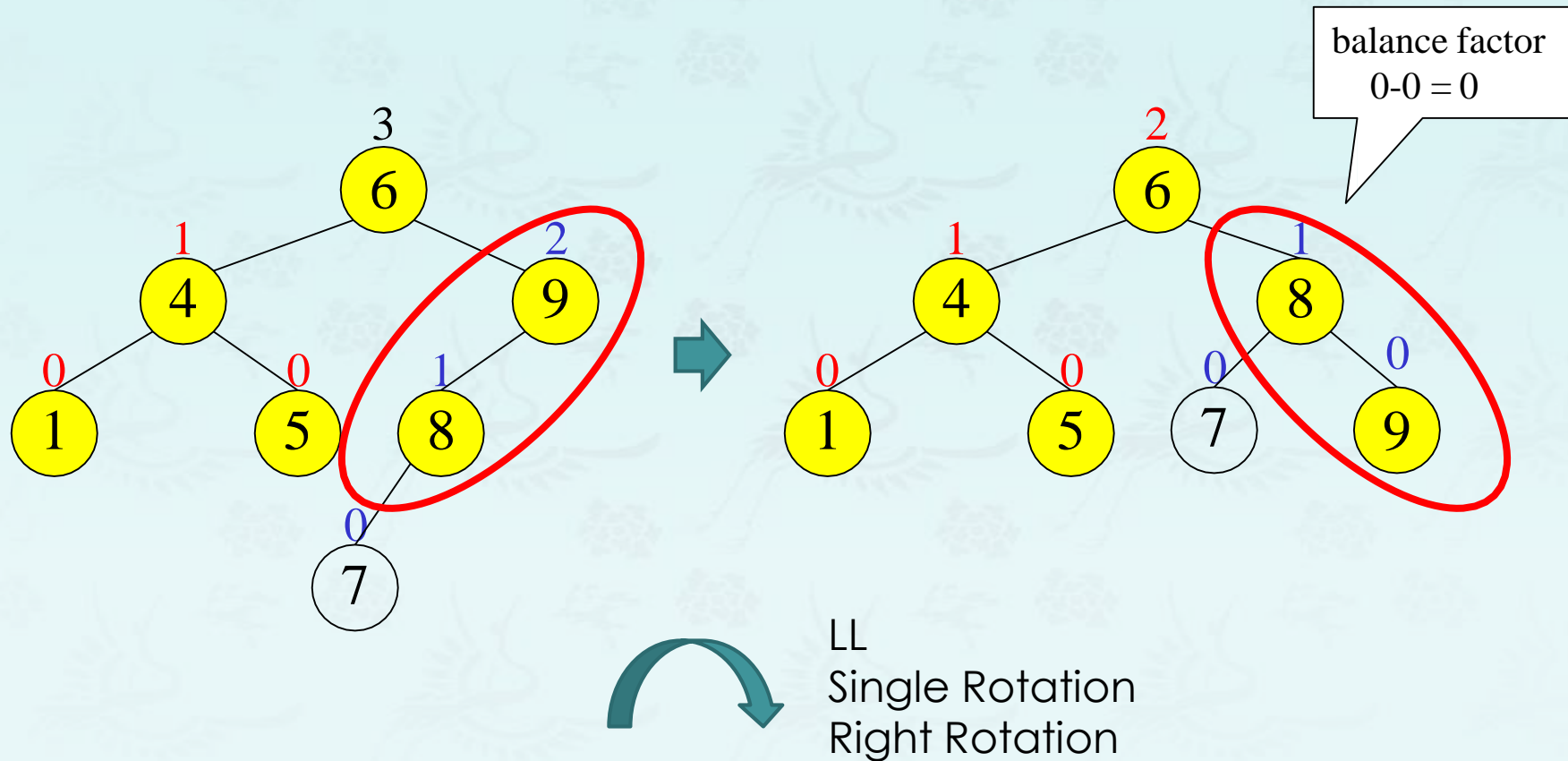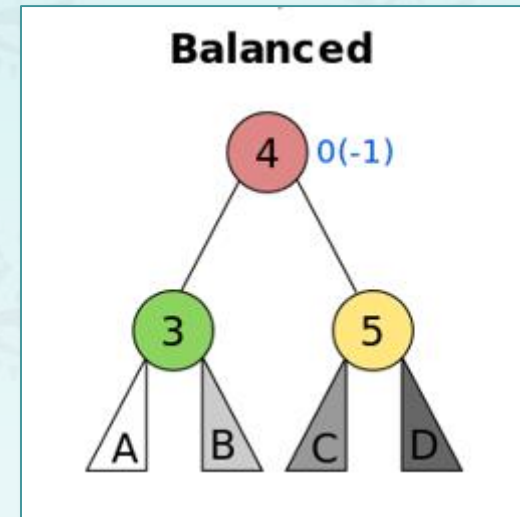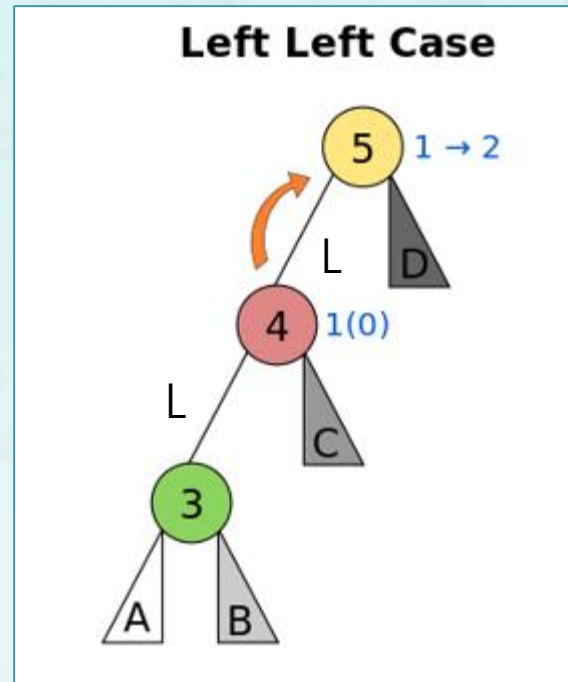
LL Case
Single Right Rotation

AVL Tree Balanced?

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

24

AVL Tree Balanced?

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

25

AVL Tree Balanced?

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

26

Single Rotation in an AVL Tree

Right Right Case

-1 → -2

3

A

R

-1(0)

4

B

R

5

C D

Balanced

0(1)

4

3

5

A B C D
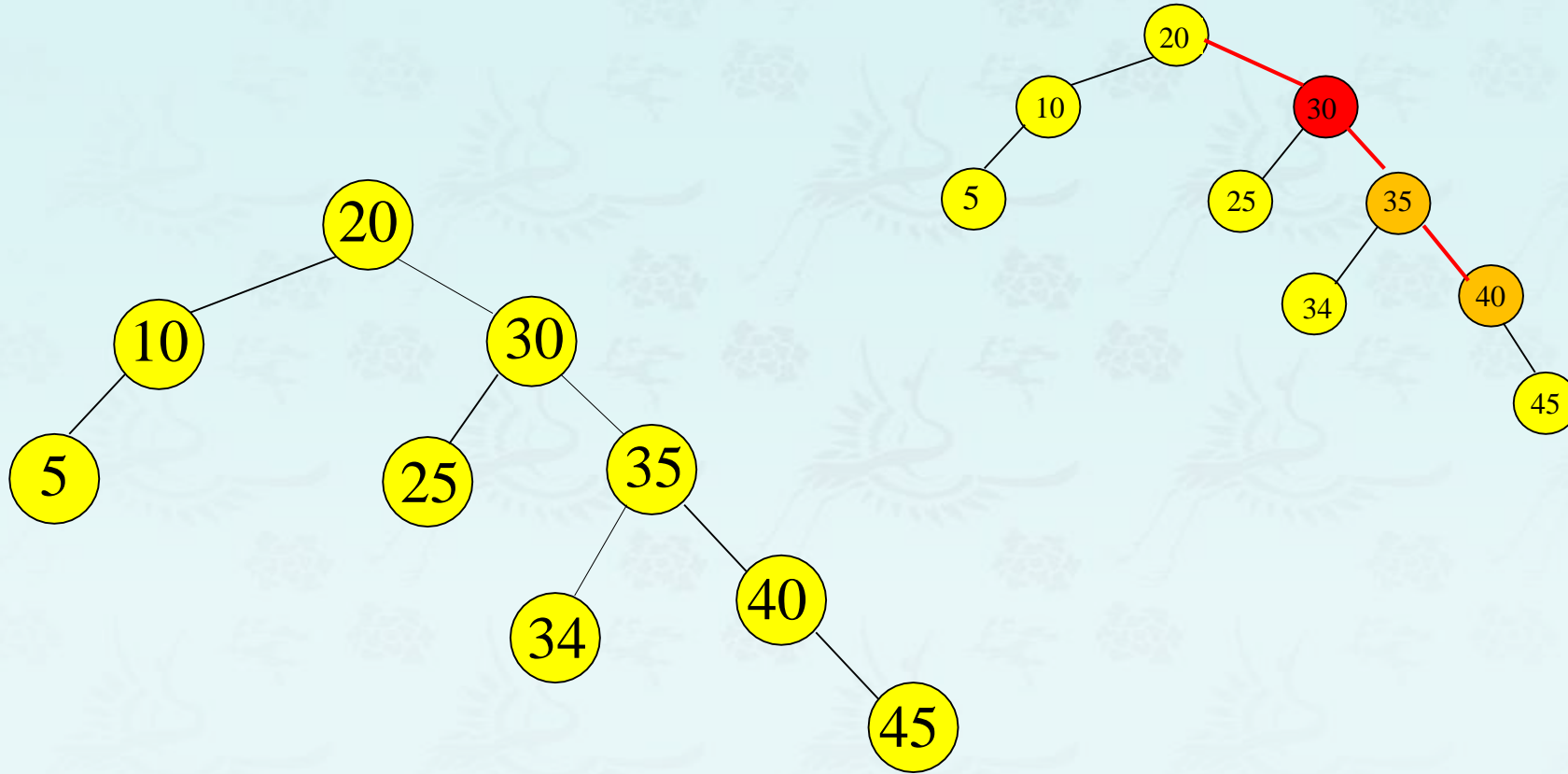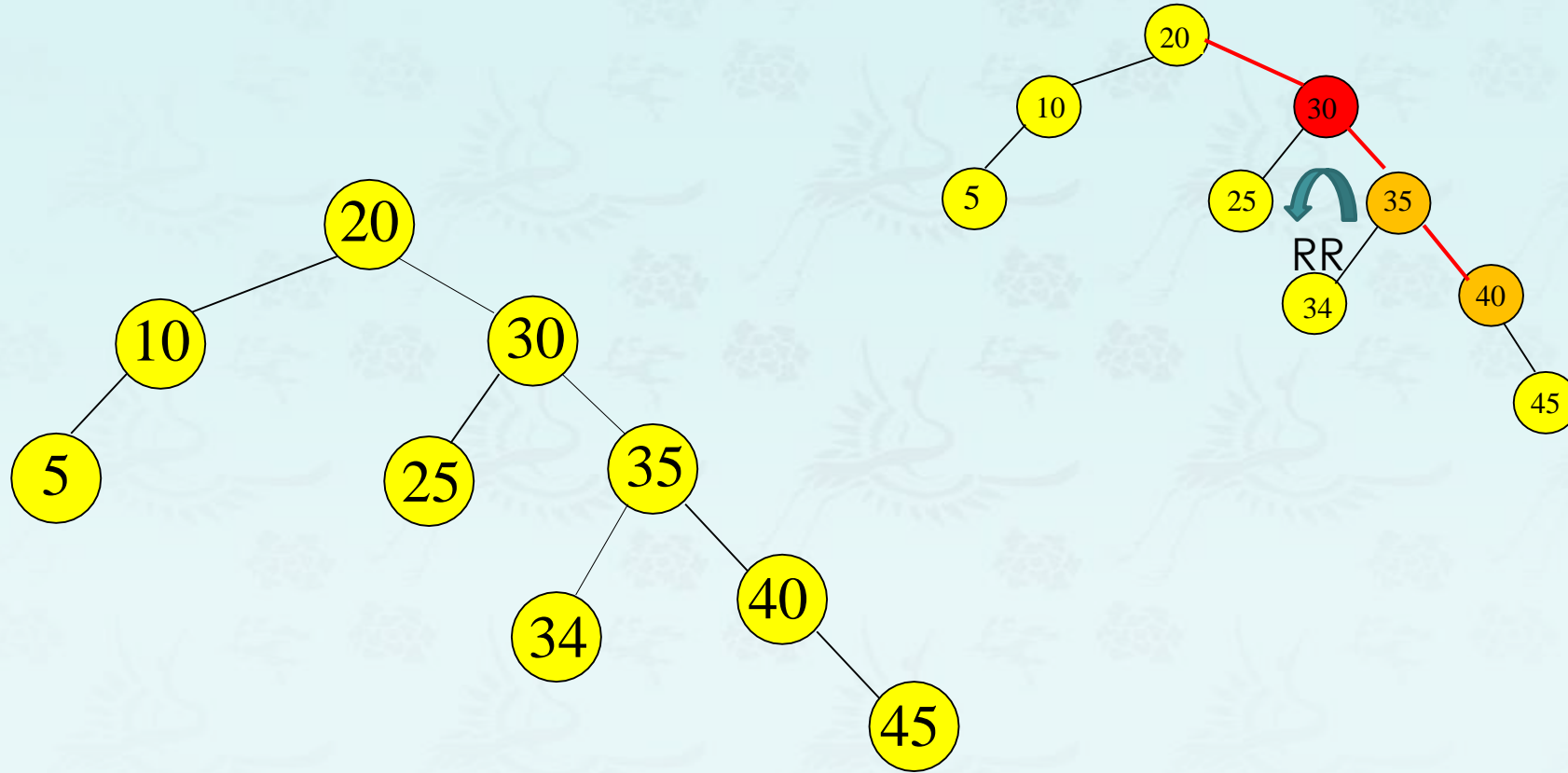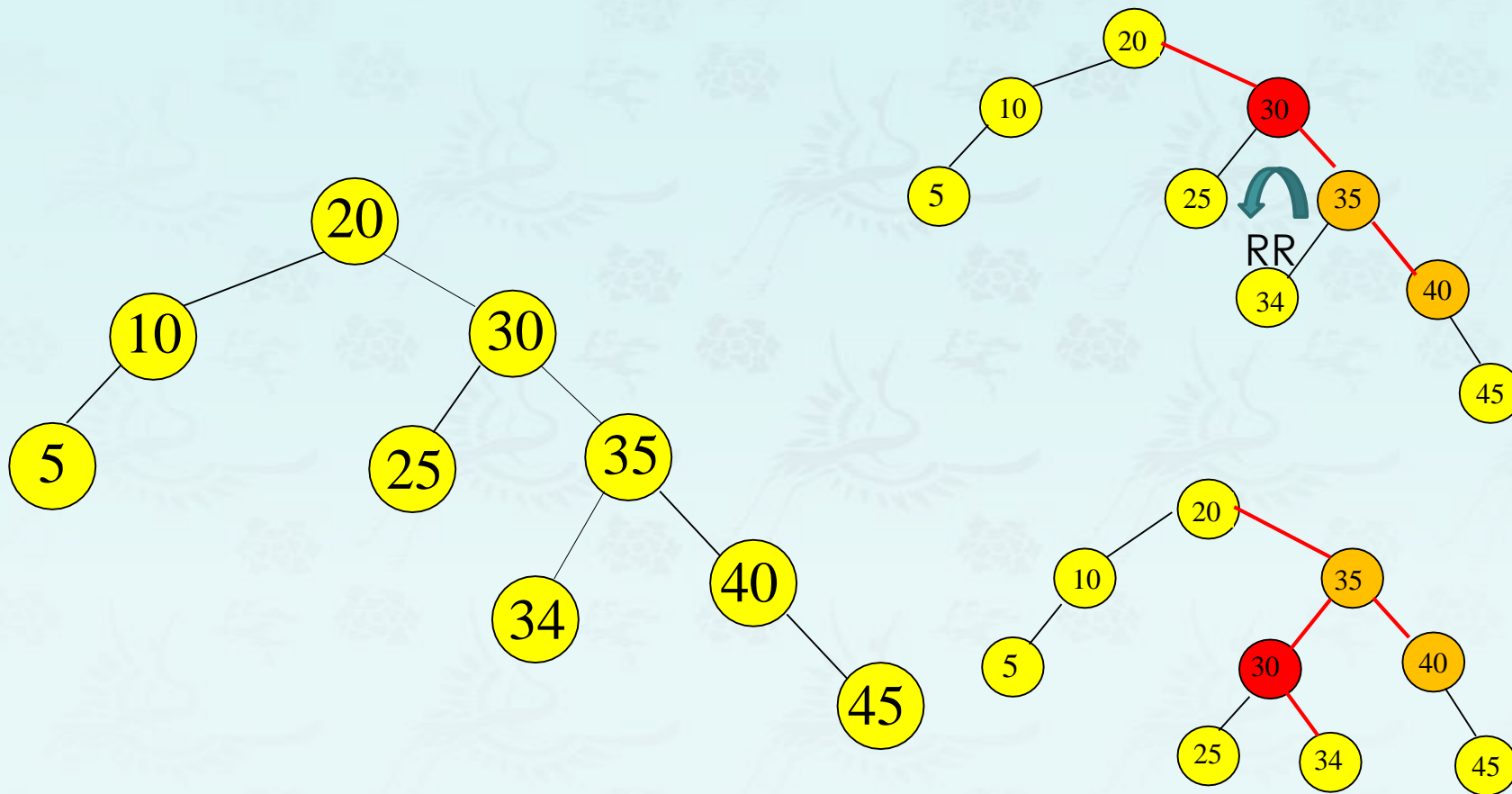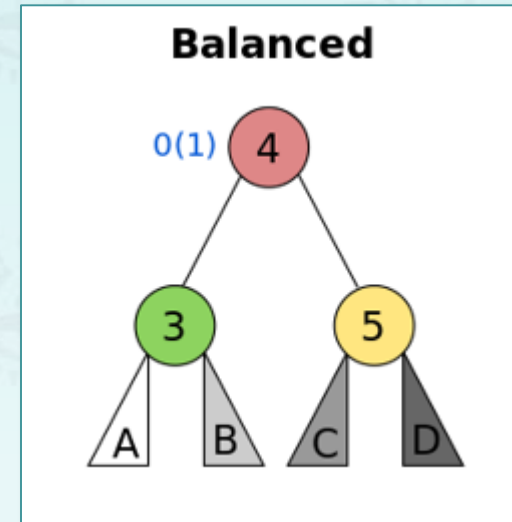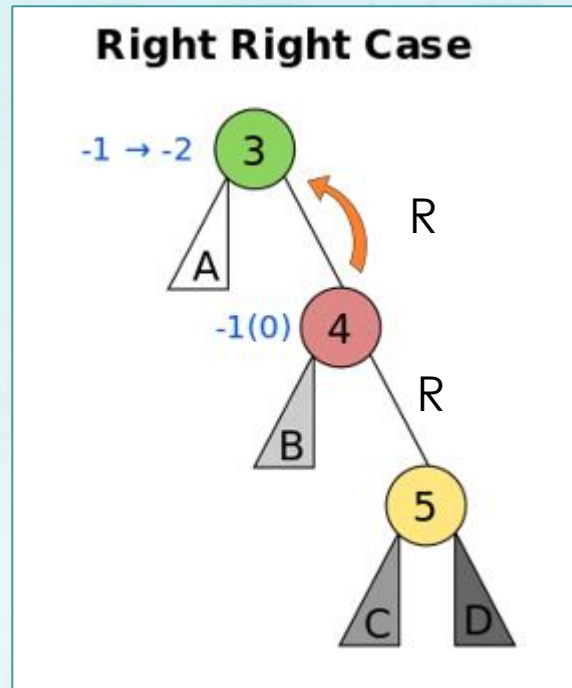
RR Case
Single Left Rotation

AVL Tree Balanced?
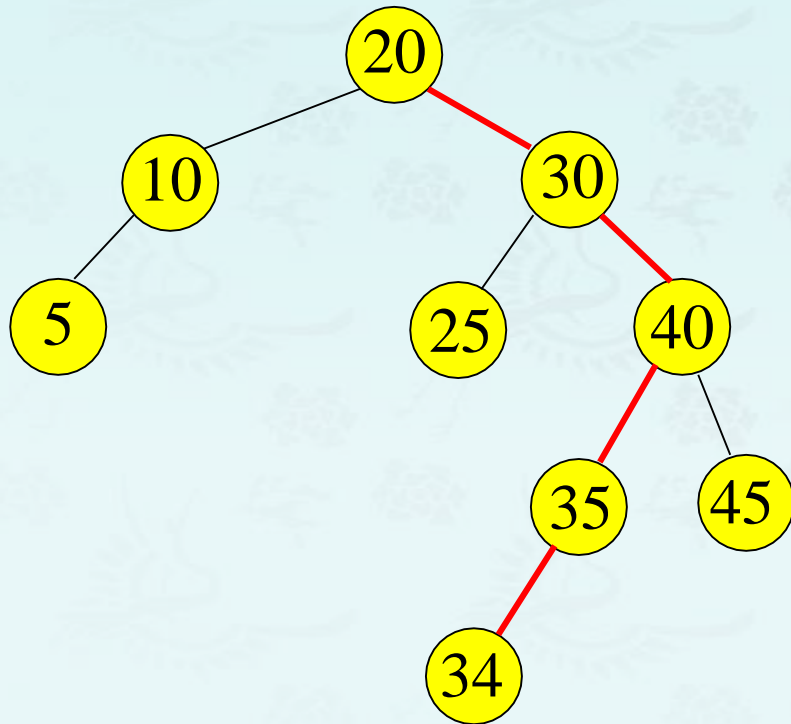
Insertion of 34
Imbalance at 30
Balance factor at 30 = -2

AVL Tree Balanced?
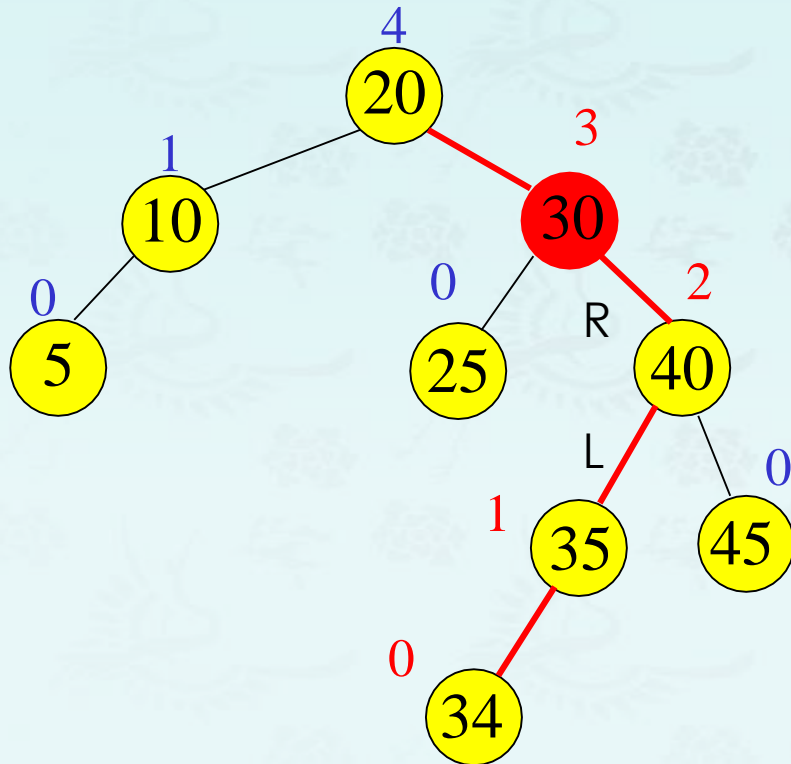
Insertion of 34
Imbalance at 30
Balance factor at 30 = -2

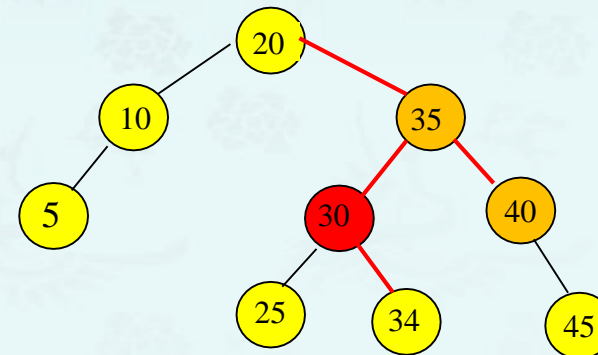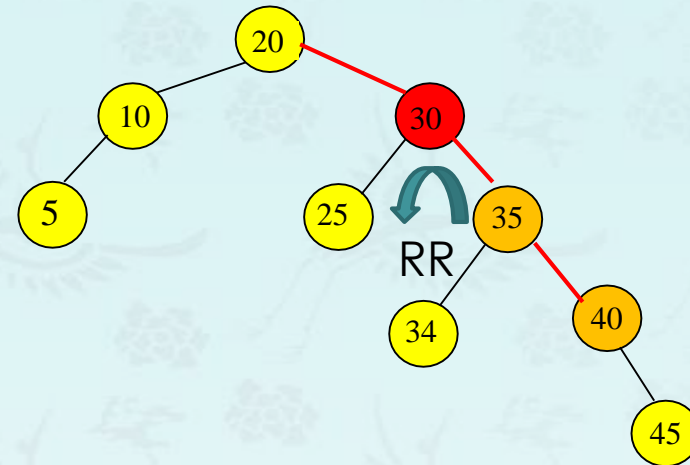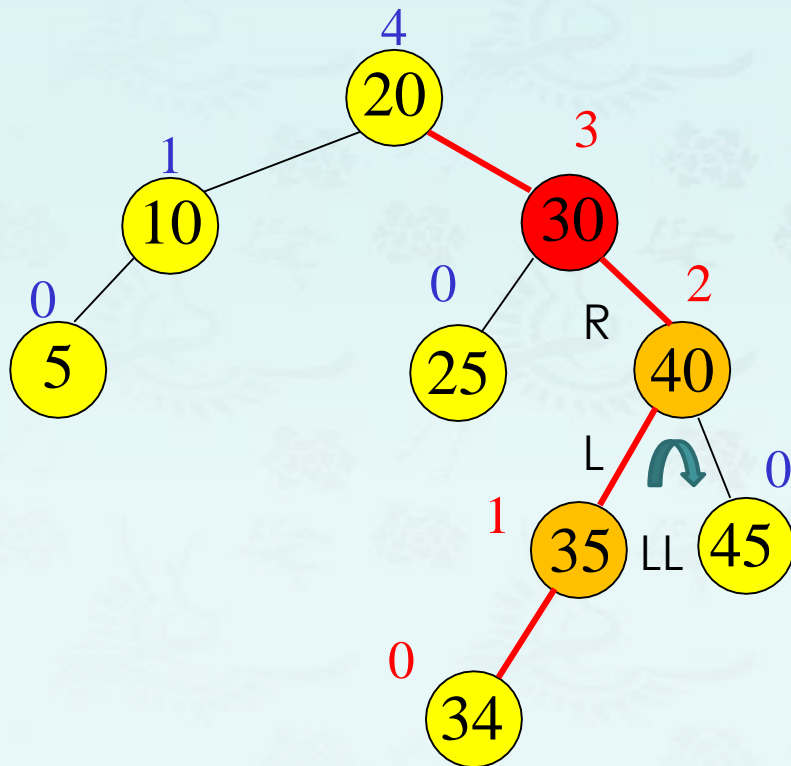Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

33

Double rotation RL

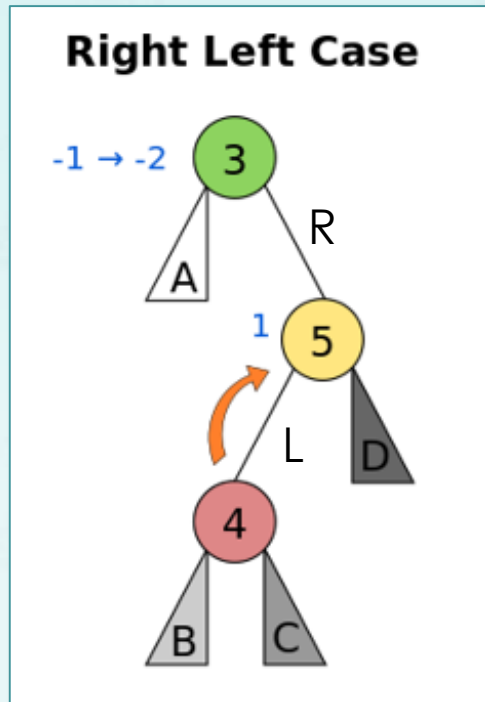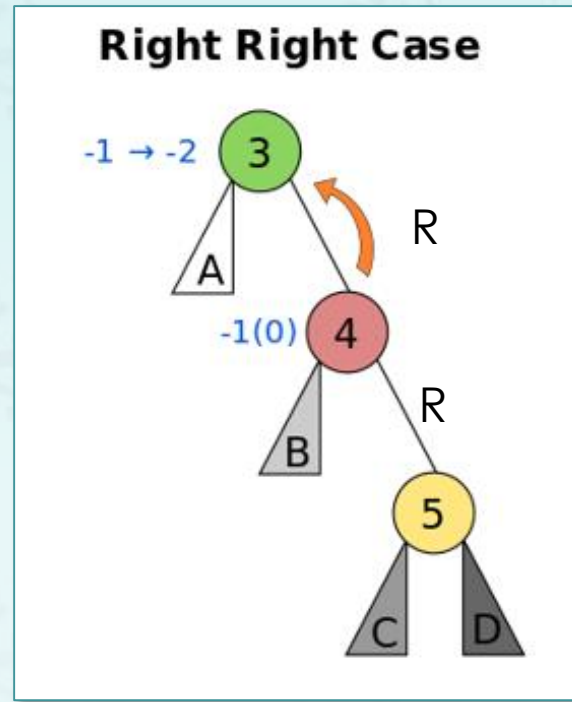**Insertion of 34**
Imbalance at 30
Balance factor at 30 = -2

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

34

Double rotation RL

Insertion of 34
Imbalance at 30
Balance factor at 30 = -2

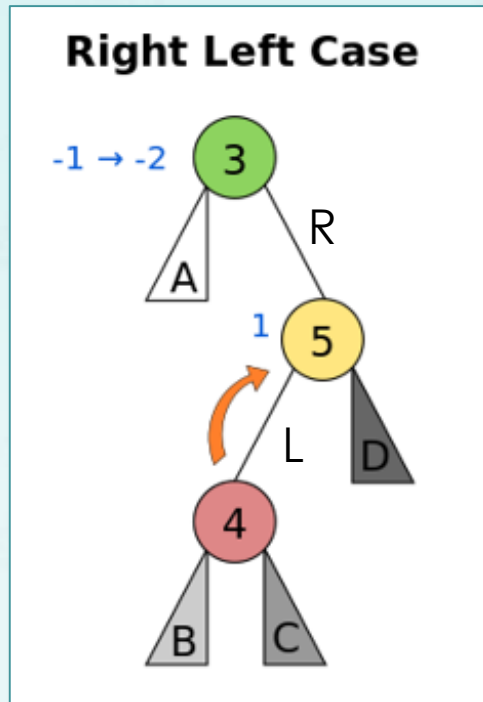Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

35

Double rotation RL

Insertion of 34
Imbalance at 30
Balance factor at 30 = -2

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

36
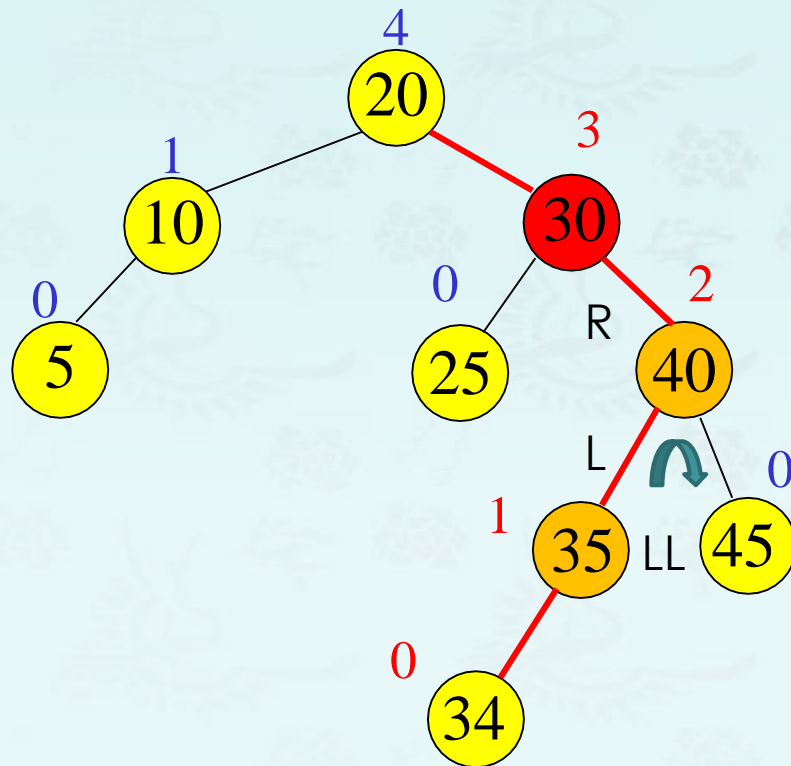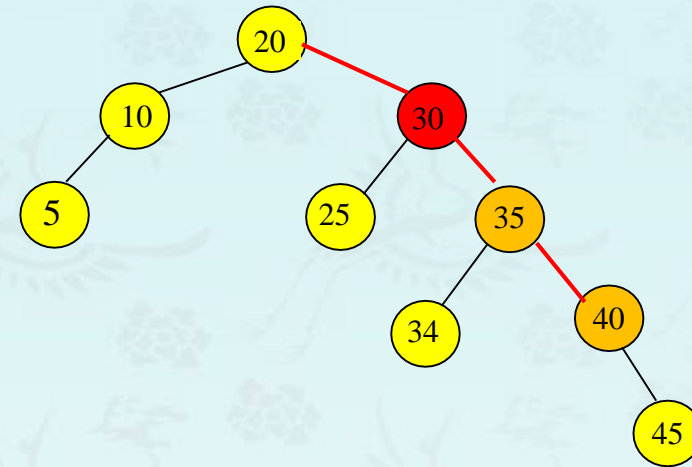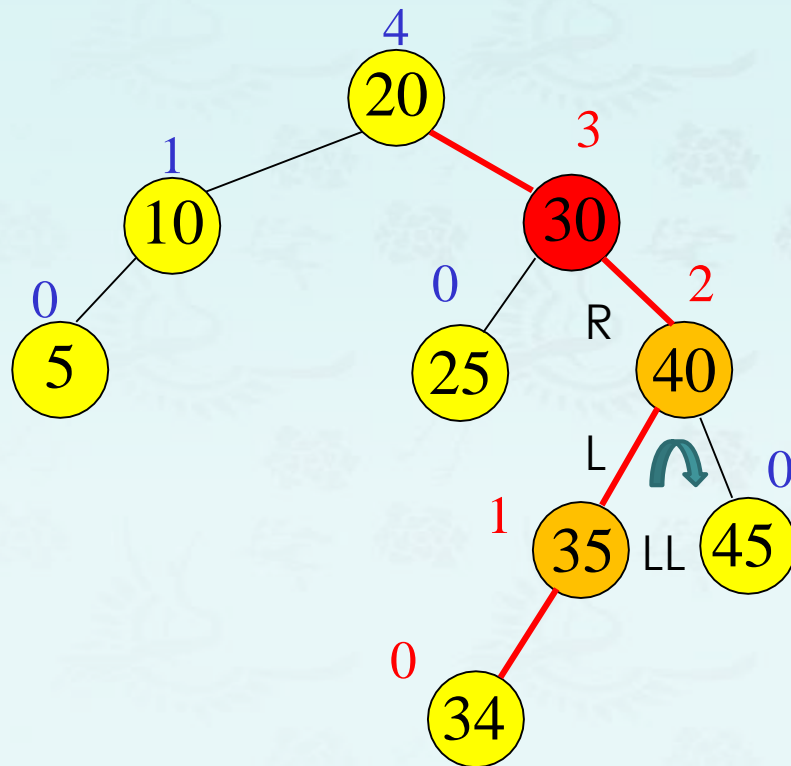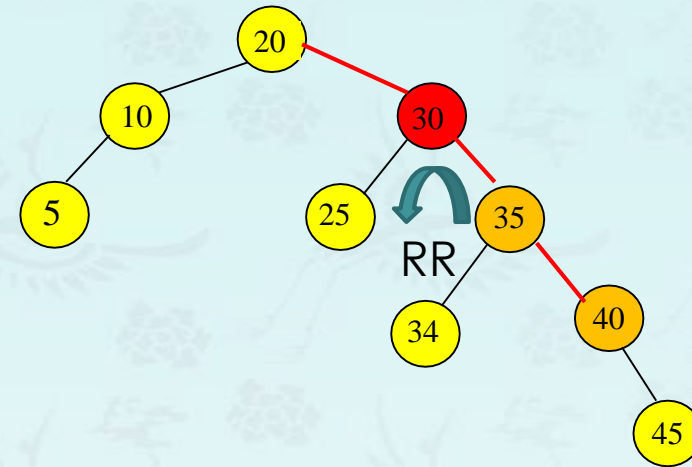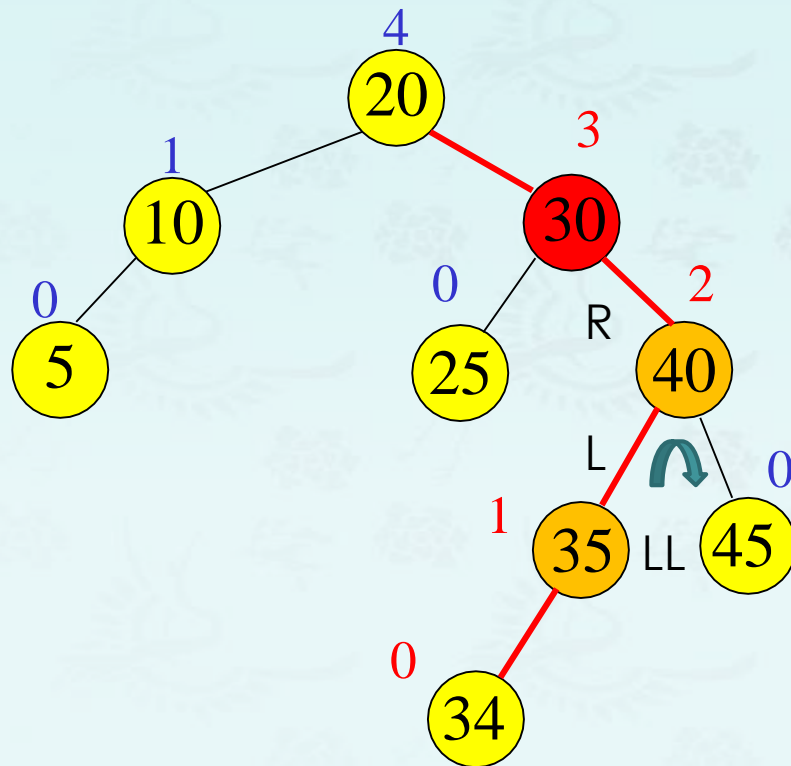
Double rotation RL

Insertion of 34
Imbalance at 30
Balance factor at 30 = -2

RL
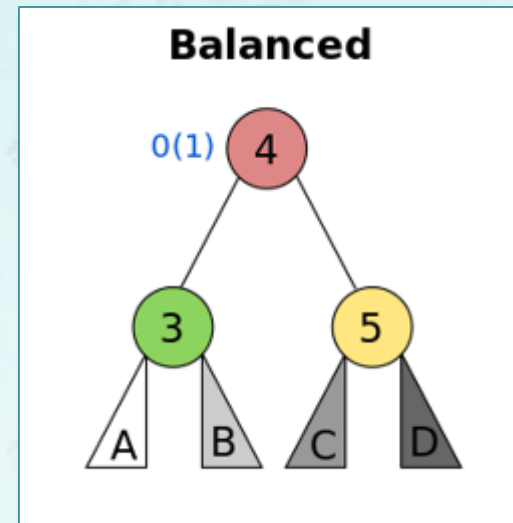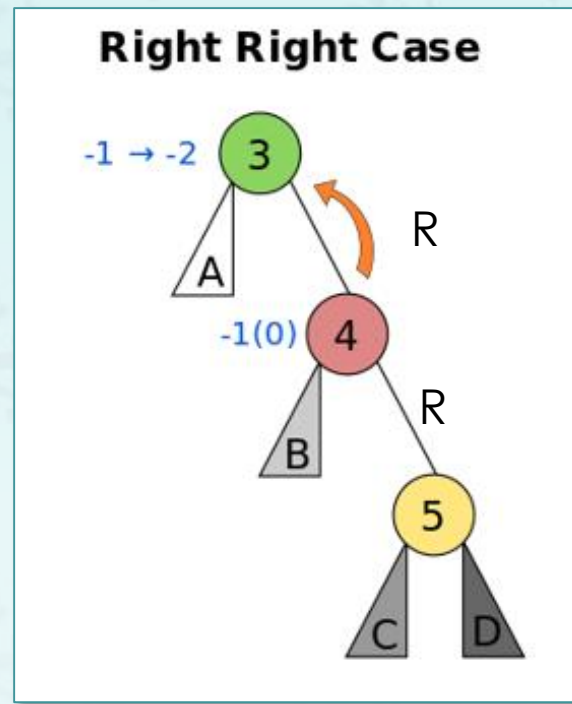**double rotation**
LL rotation + RR rotation

Double rotation – RL Case
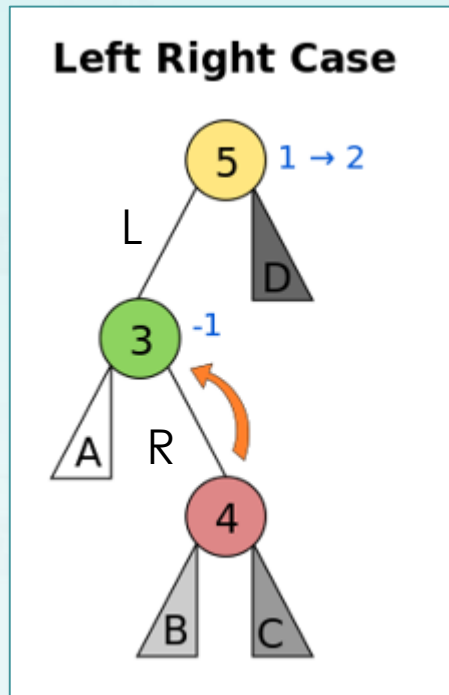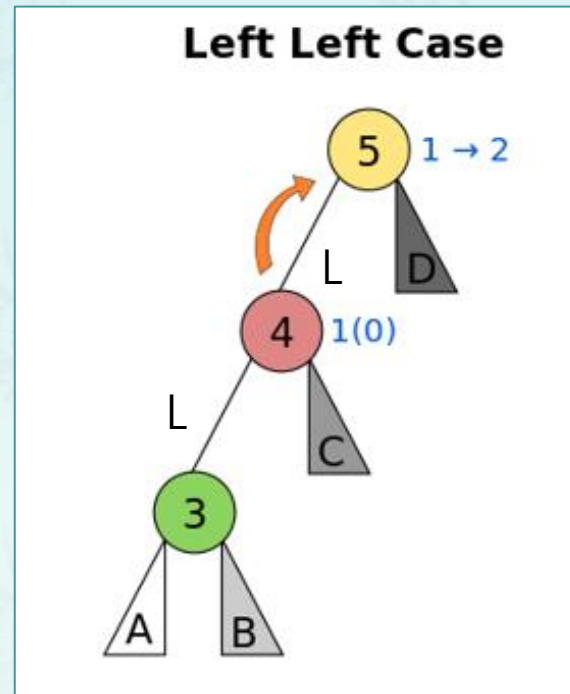

Right Left Case

$-1 \to -2$

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

38

Double rotation – RL Case

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

39

Double rotation – RL Case

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

40

Double rotation – LR Case

Left Right Case

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

41

Double rotation – LR Case

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

42

Double rotation – LR Case

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

43

Insertions in AVL Trees

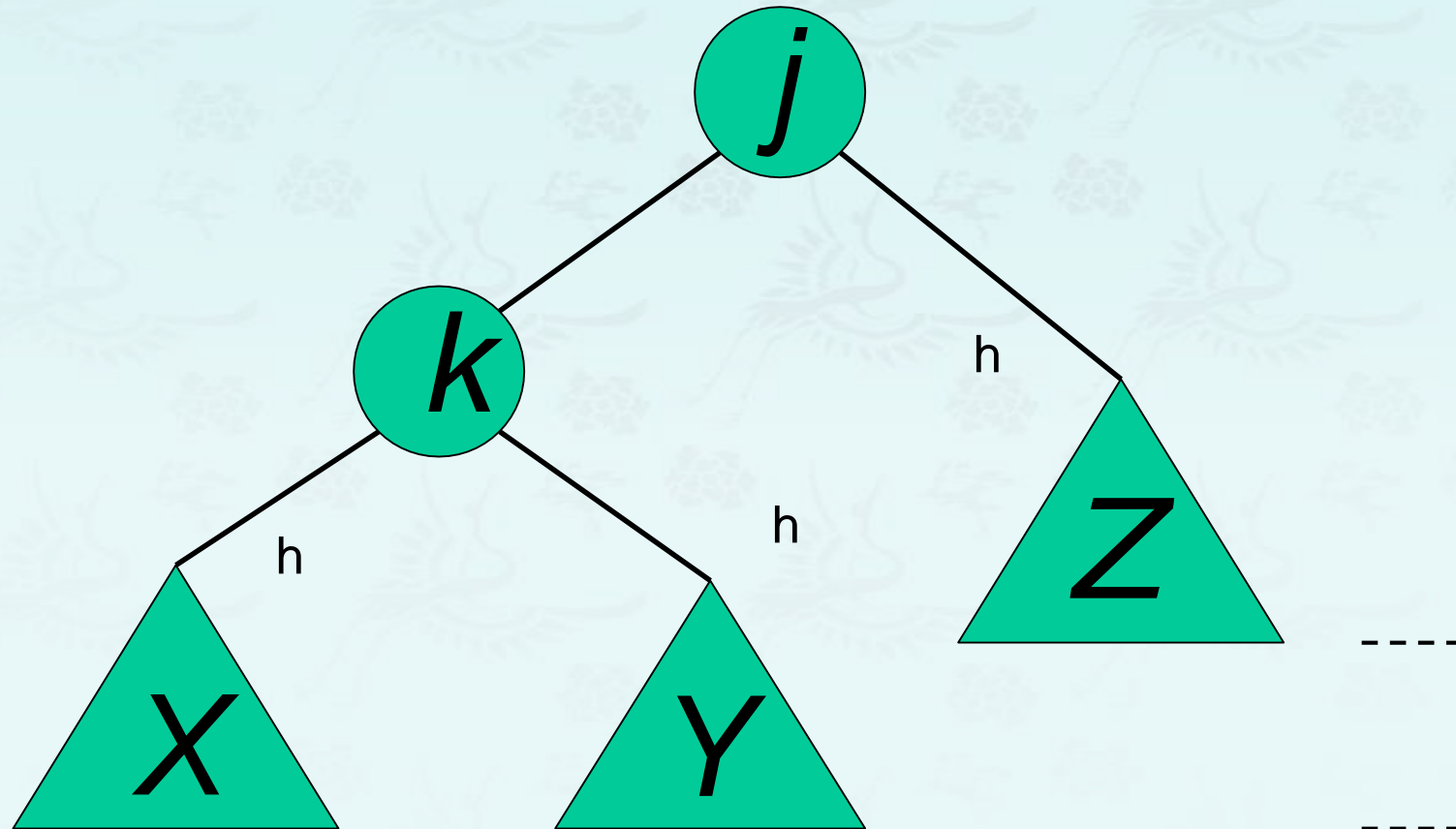Let the node that needs rebalancing be a.

There are 4 cases:

- Outside Cases (require single rotation) :
    1. Insertion into left subtree of left child of a.
    2. Insertion into right subtree of right child of a.

- Inside Cases (require double rotation) :
    1. Insertion into right subtree of left child of a.
    2. Insertion into left subtree of right child of a.

The rebalancing is performed through four separate rotation algorithms.

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University
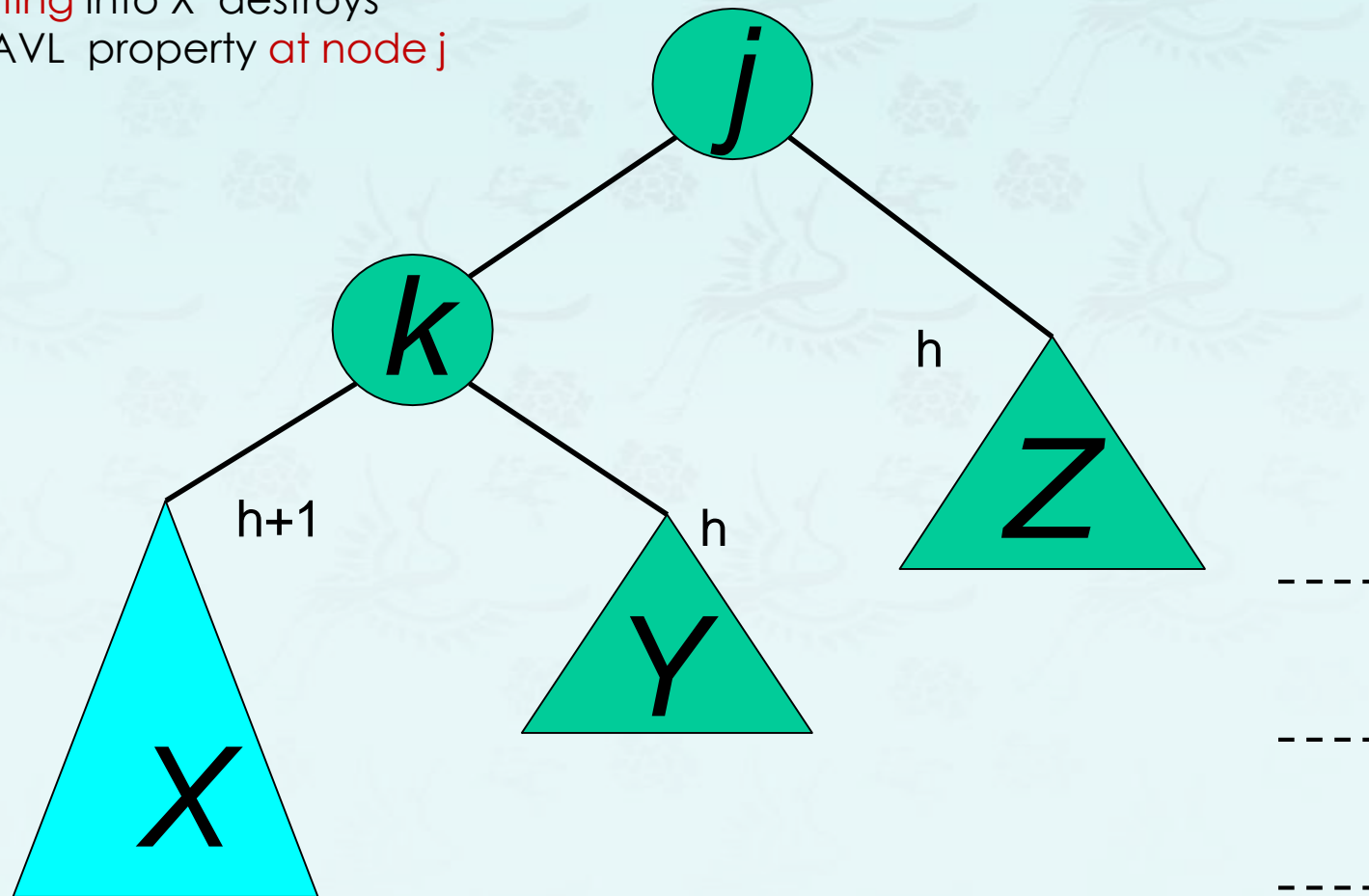
44

AVL Insertion: Outside Case

Consider a valid AVL subtree

AVL Insertion: Outside Case

Consider a valid AVL subtree

Inserting into X destroys
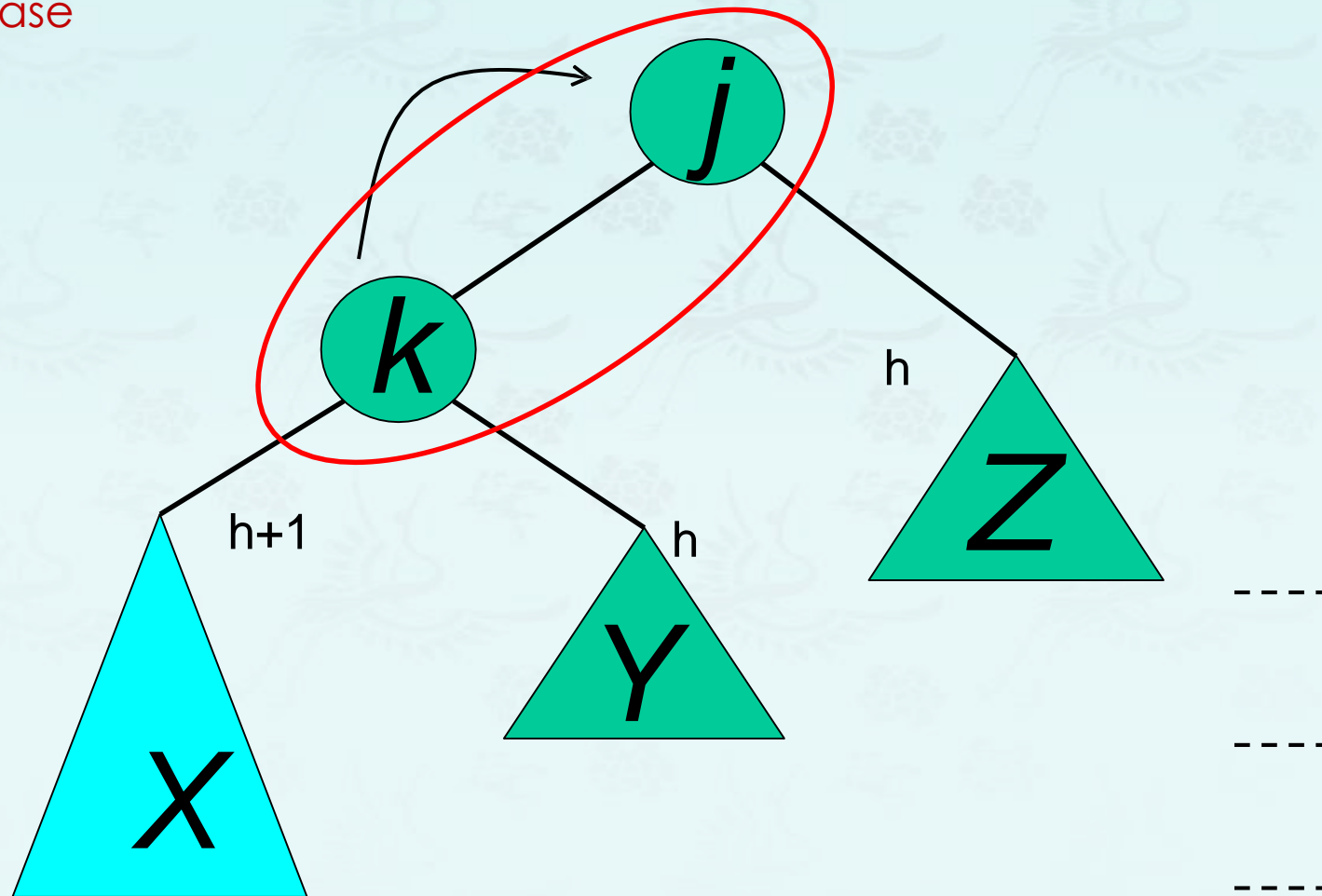the AVL property at node j

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University
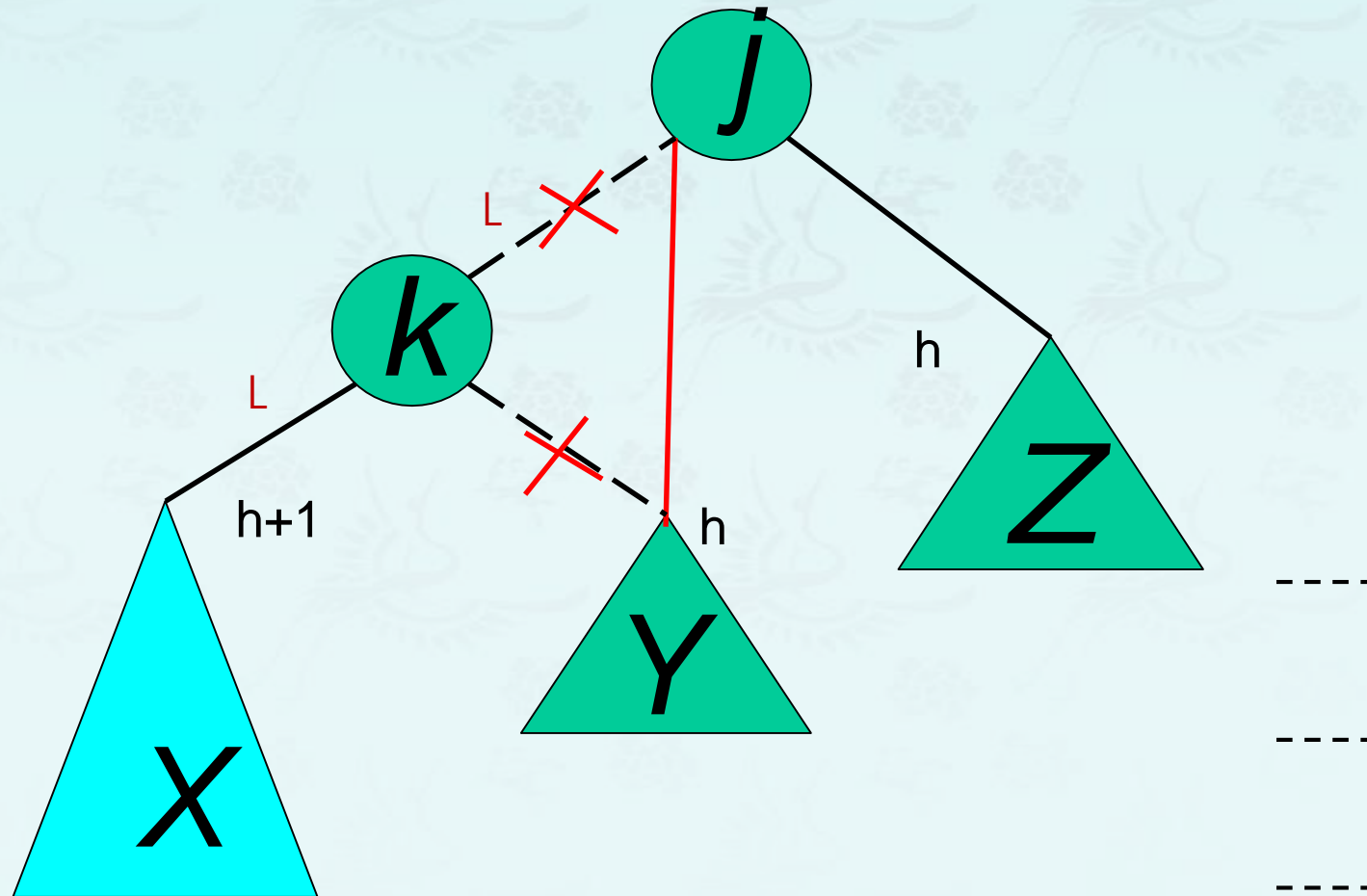
46

AVL Insertion: Outside Case

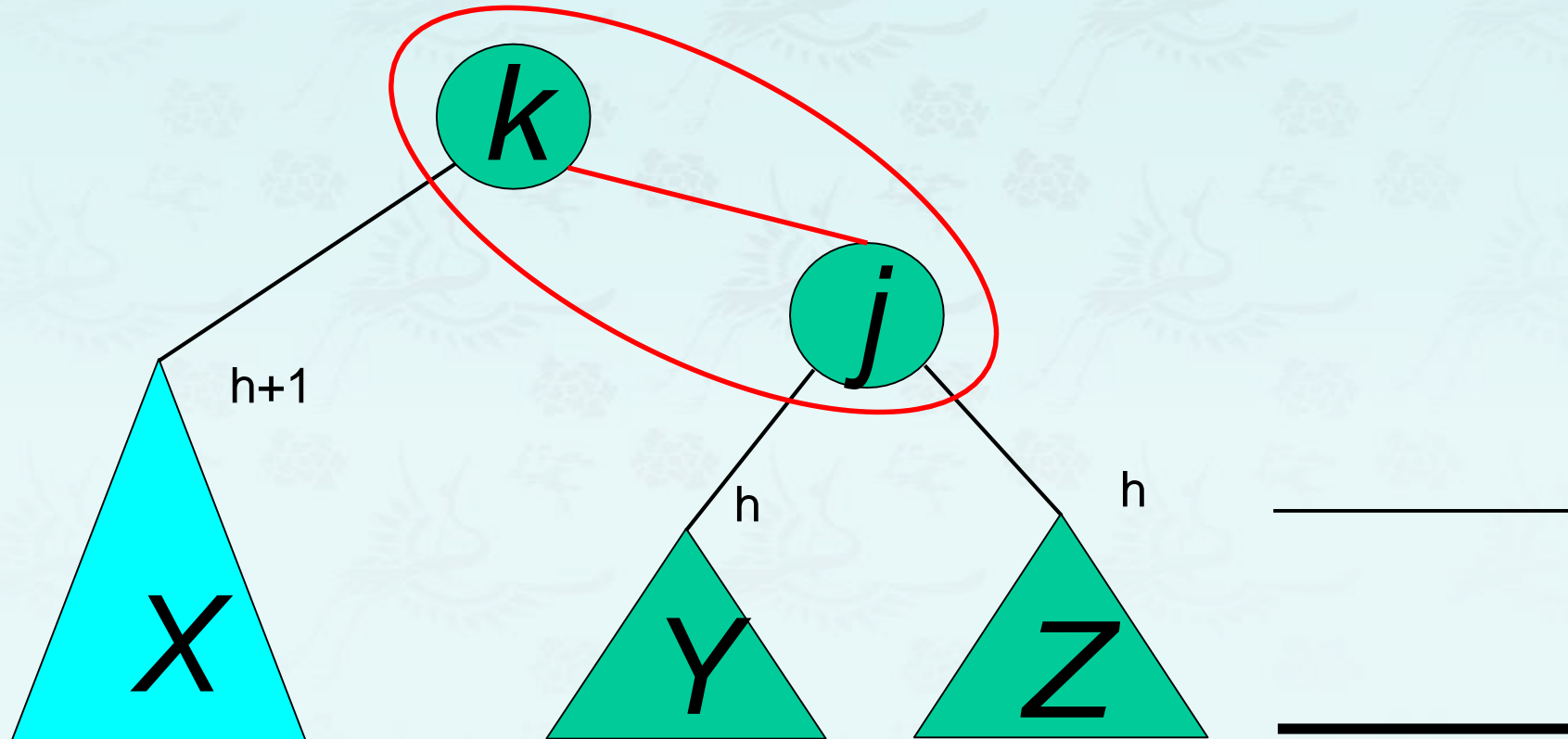Do a "right rotation"
LL Case

Single right rotation

Do a "right rotation"

Outside Case Completed

AVL property has been restored!
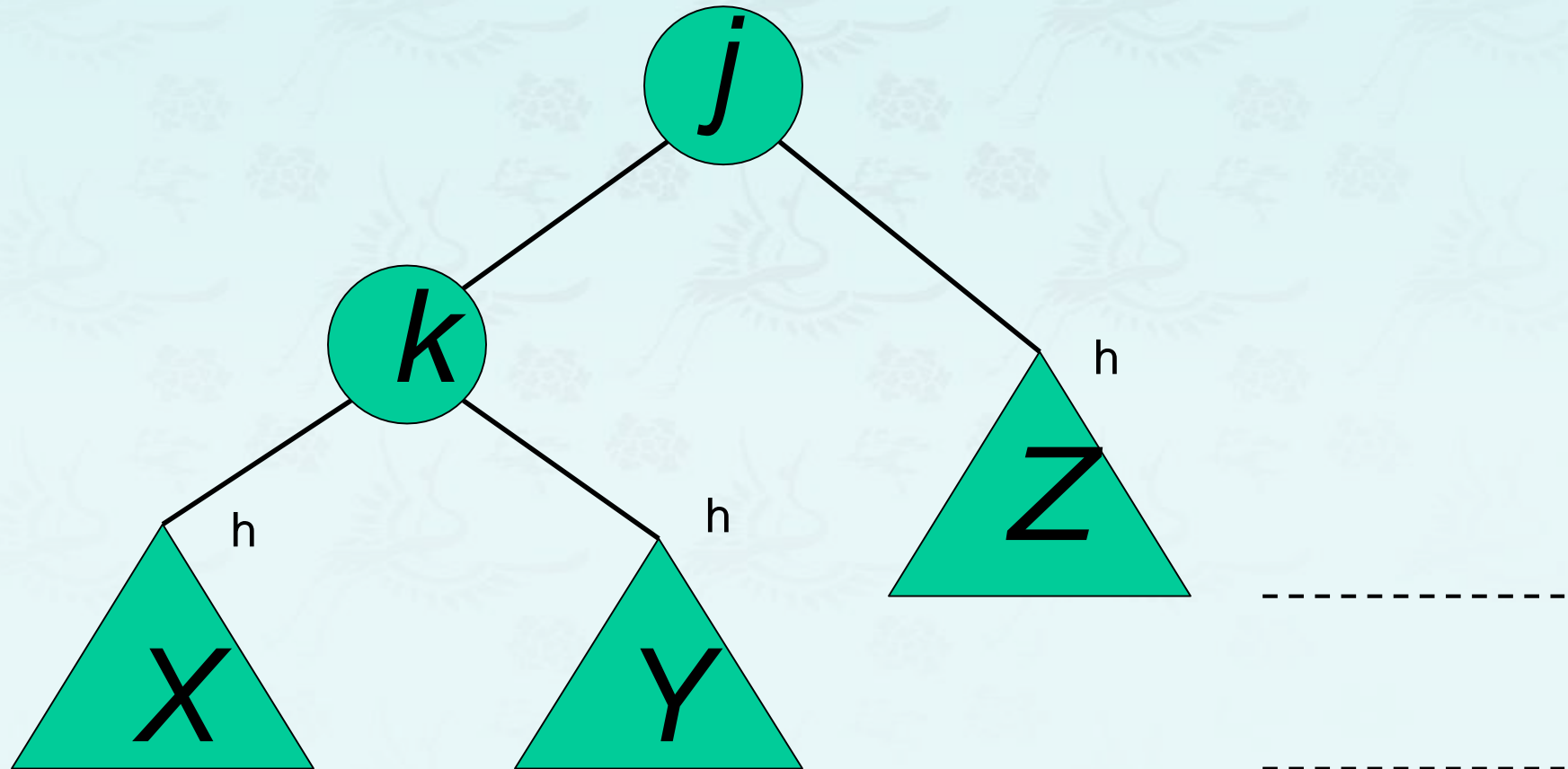
LL Case – Single Rotation

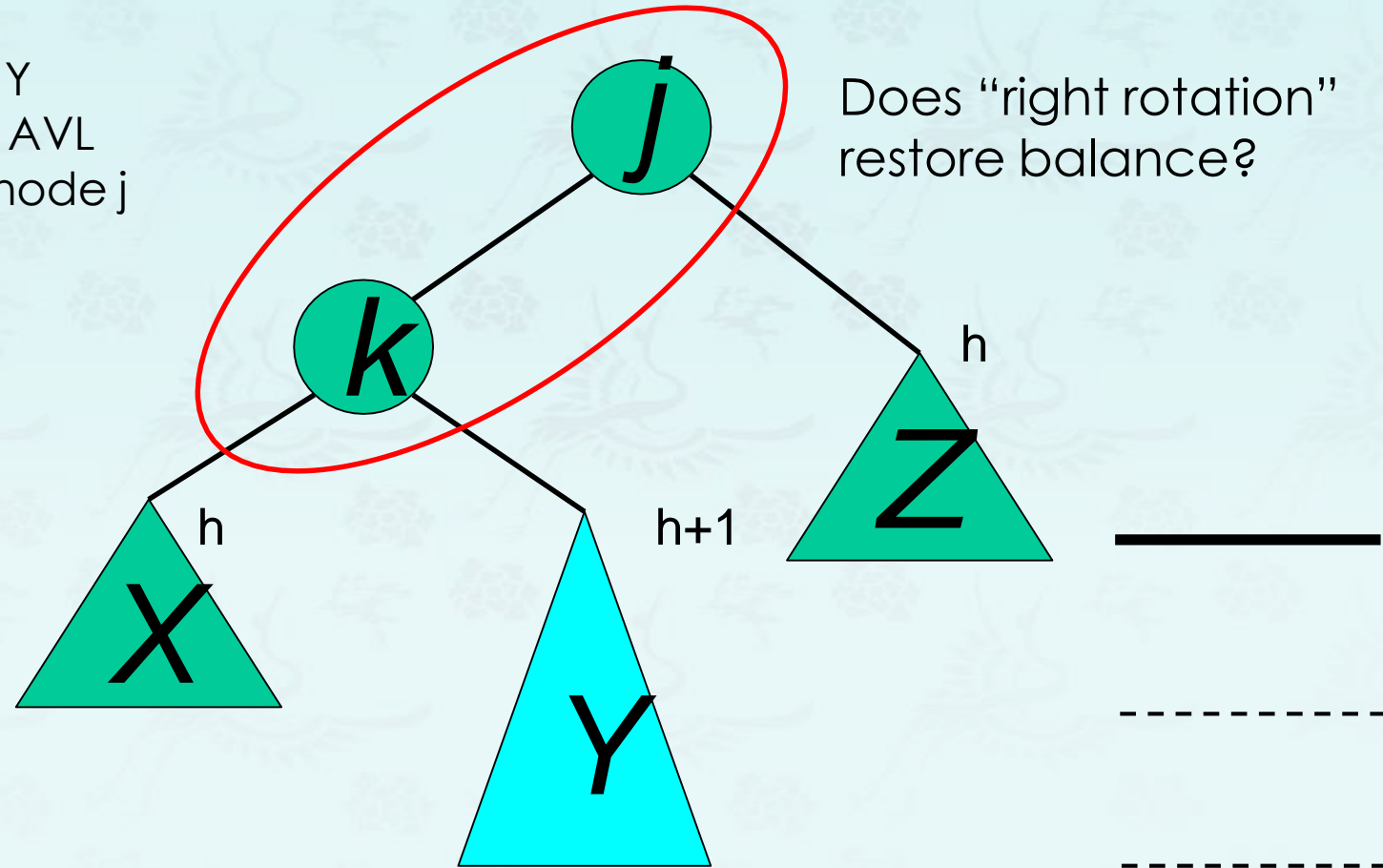"Right rotation" done!
("Left rotation" is mirror symmetric)

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

49

AVL Insertion: Inside Case

Consider a valid AVL subtree

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

50

AVL Insertion: Inside Case

Inserting into Y destroys the AVL property at node j

Does "right rotation" restore balance?

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

51

AVL Insertion: Inside Case

"Right rotation" does **not** restore balance…
now k is out of balance

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

52
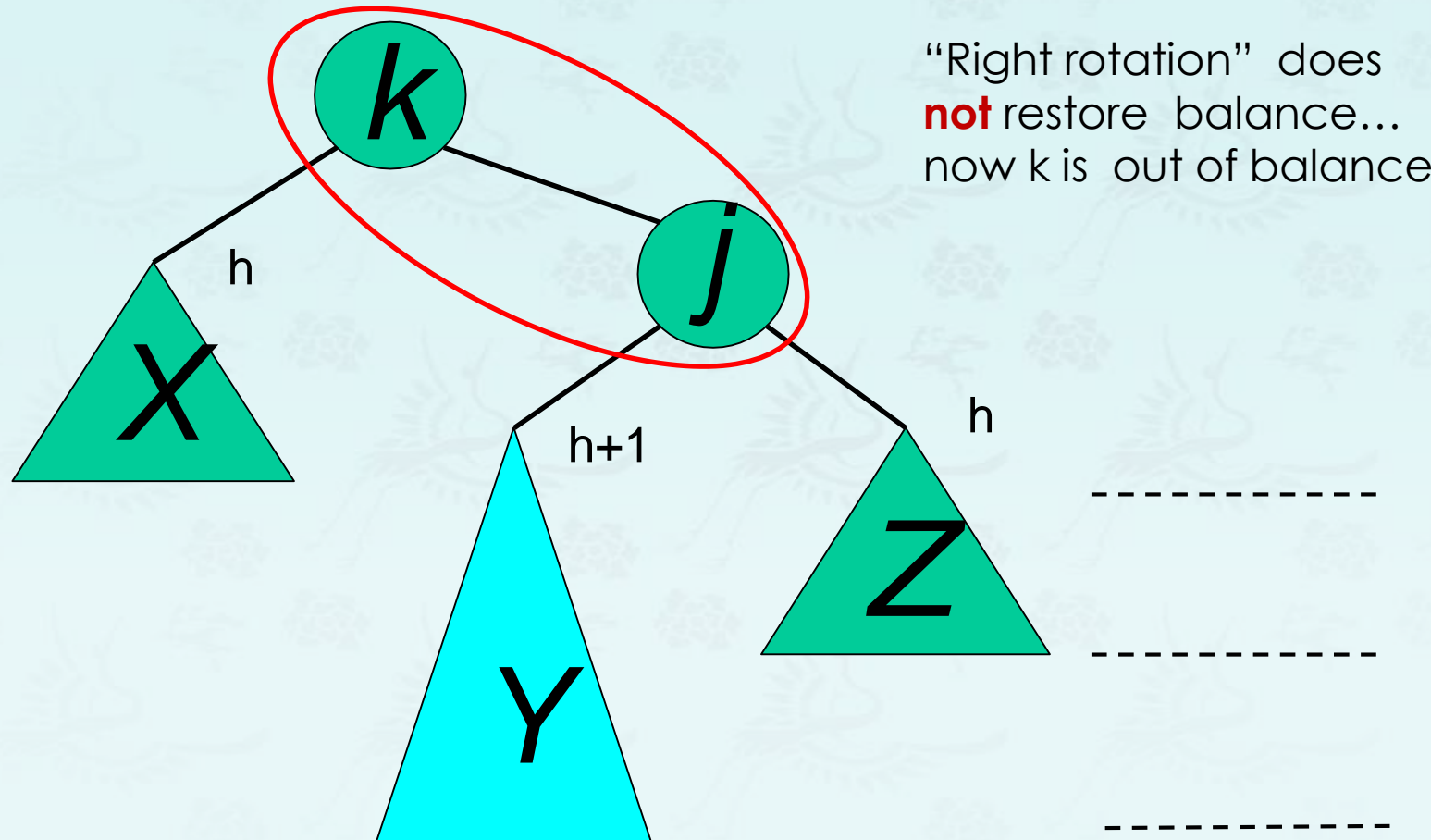
AVL Insertion: Inside Case

Y = node i and
subtrees V and W

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

53

AVL Insertion: Inside Case

We will do a left-right " double rotation" . . .

LR Case

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

54
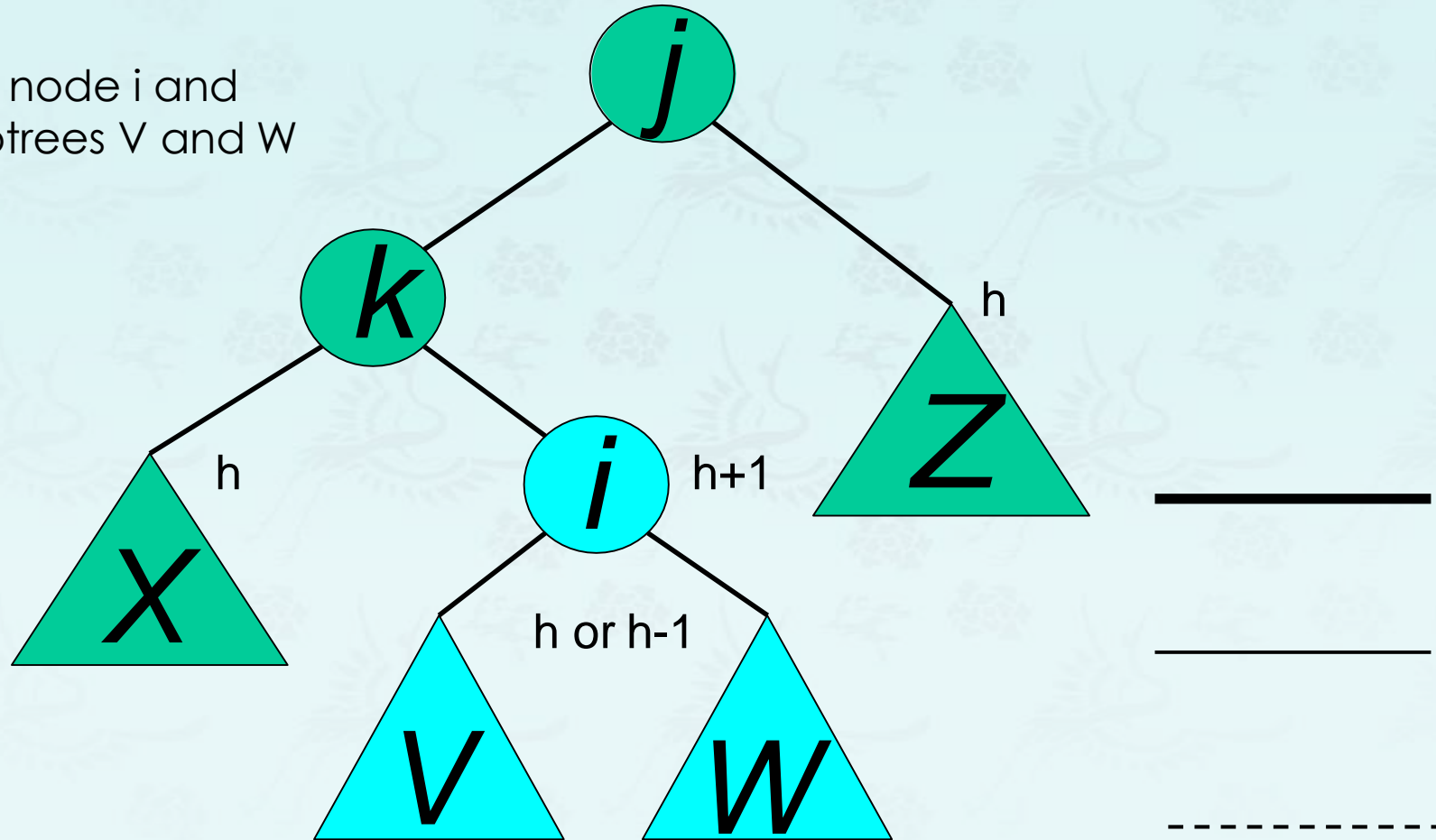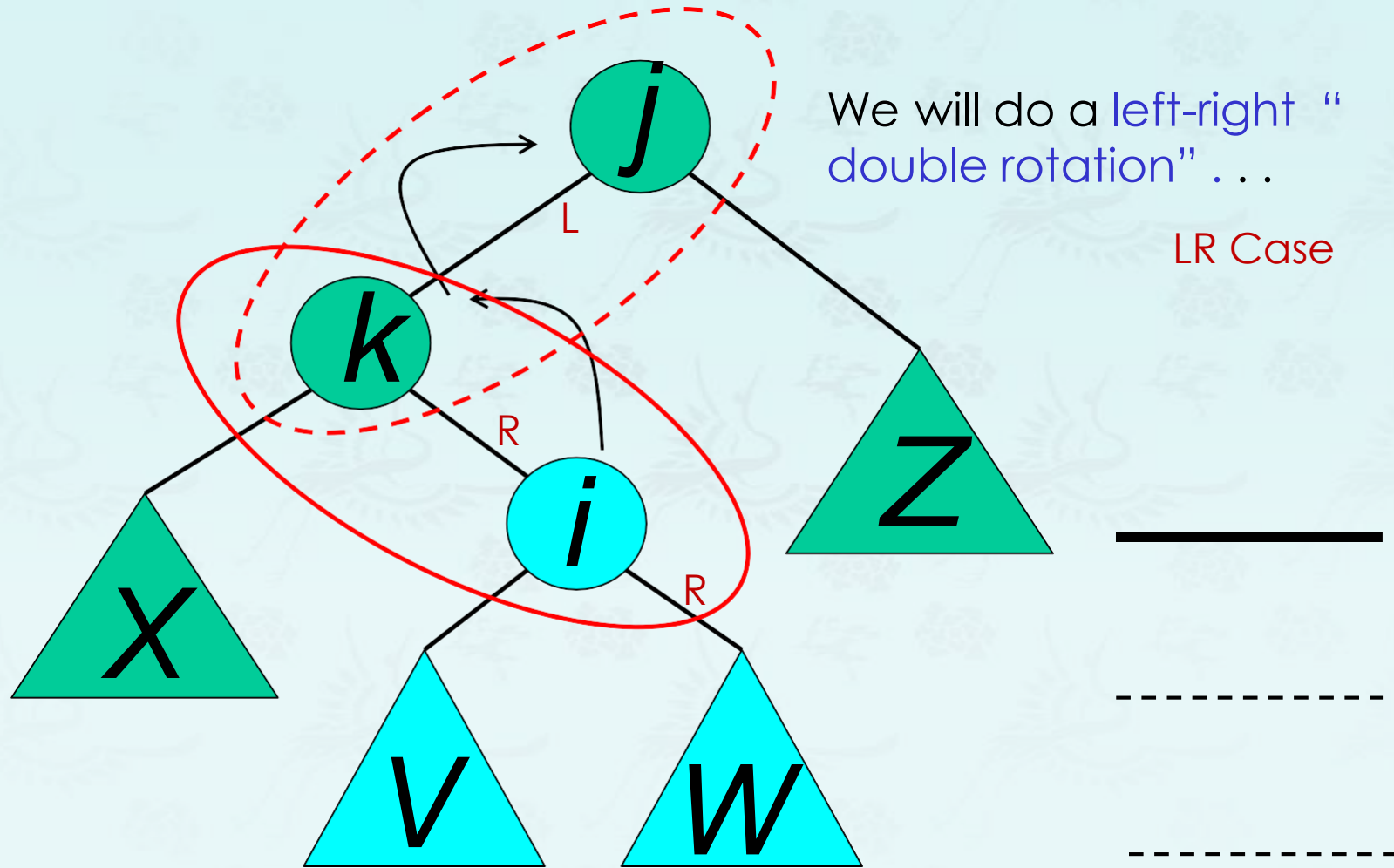
Double rotation : first rotation

left rotation complete

LR Case

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

55

Double rotation : second rotation
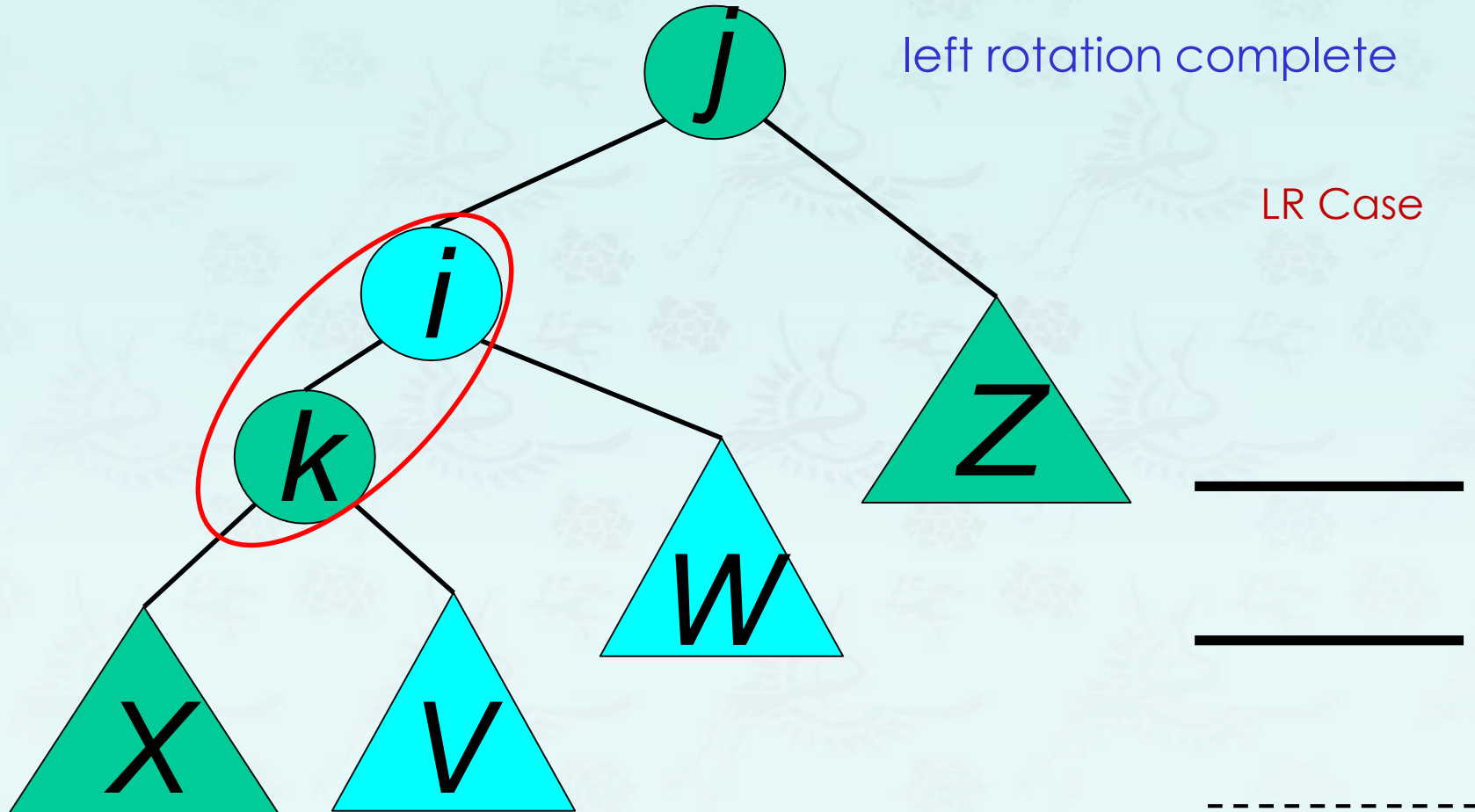
Now do a right rotation
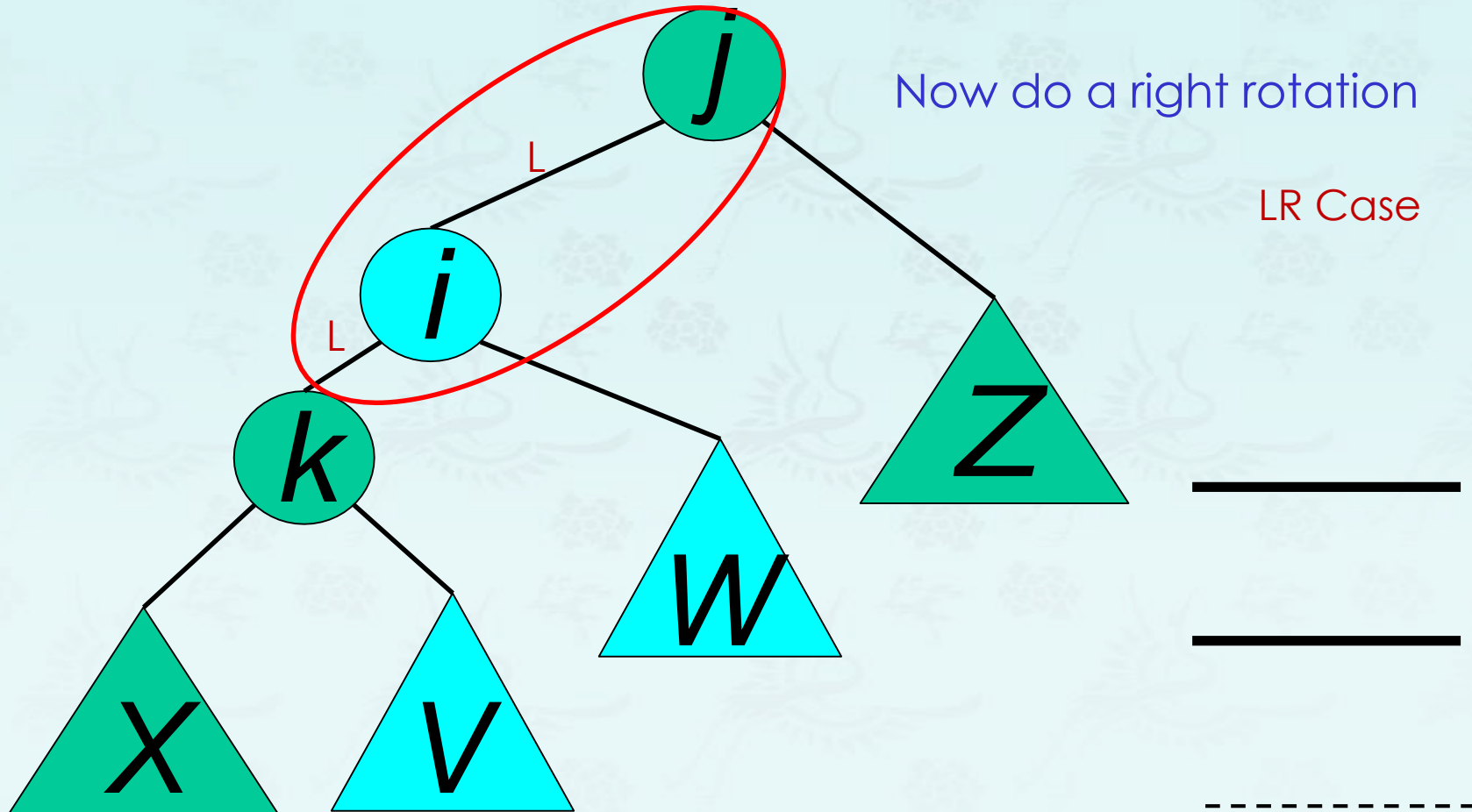
LR Case

Double rotation : second rotation

right rotation complete

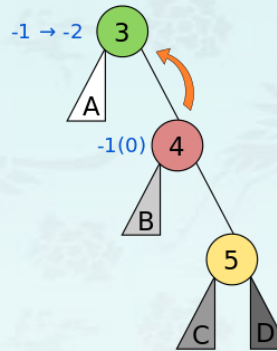Balance has been restored

LR Case
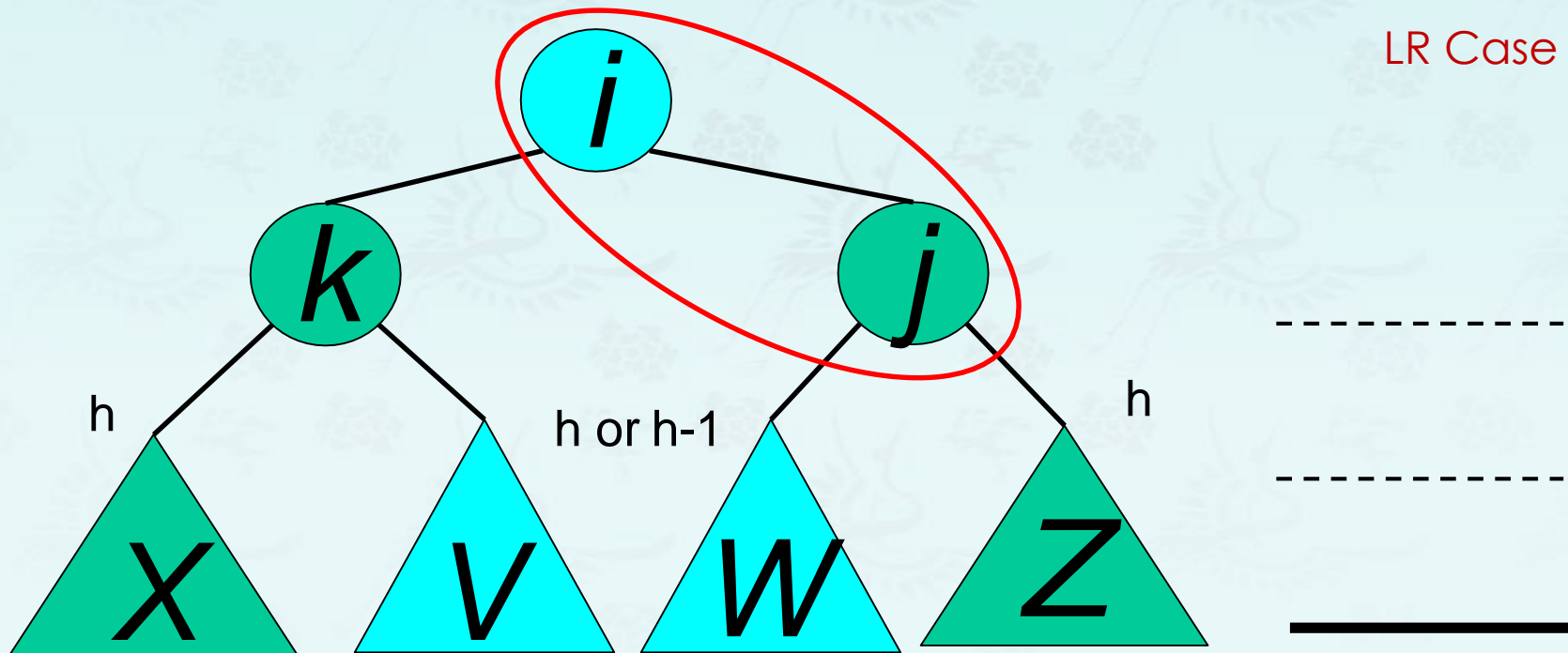
X   V   W   Z

h   h or h-1   h
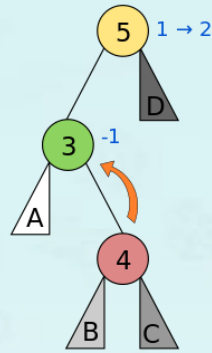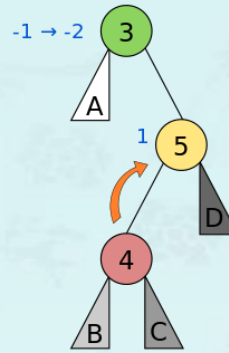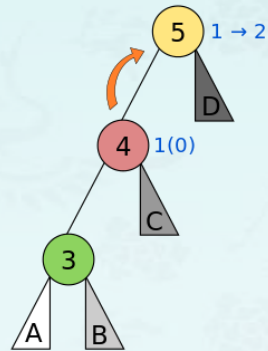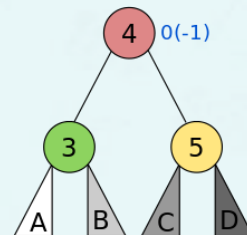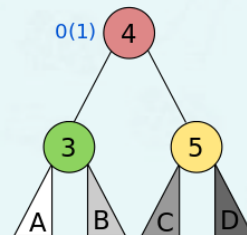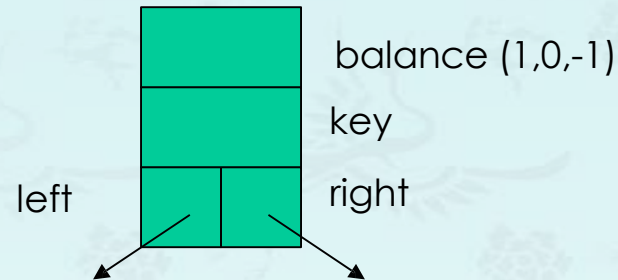
- The numbered circles represent the nodes being rebalanced.
- The lettered triangles represent subtrees which are themselves balanced AVL trees.
- A blue number next to a node denotes possible balance factors
- (those in parentheses occurring only in case of deletion).
- Source: www.wikipedia.com

58

Implementation

balance (1,0,-1)

key

left   right

- You can either keep the height or just the difference in height,
  - i.e. the balance factor; this has to be modified on the path of insertion even if you don't perform rotations

  - Once you have performed a rotation (single or double) you won't need to go back up the tree

- You may compute the balance factor on the fly after the insert is done during the recursion.
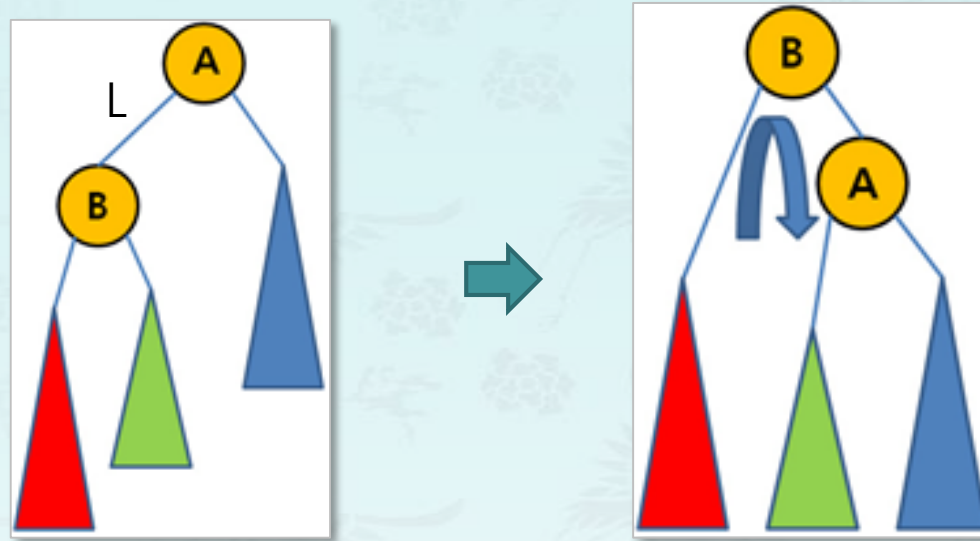
Single Rotation - LL case
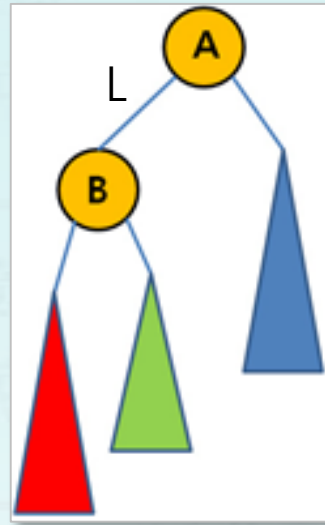
outside case


```
node rotateLL(node A)
{
    [                    ]
    [                    ]
    [                    ]
    return [          ]
}
```

Single Rotation - LL case
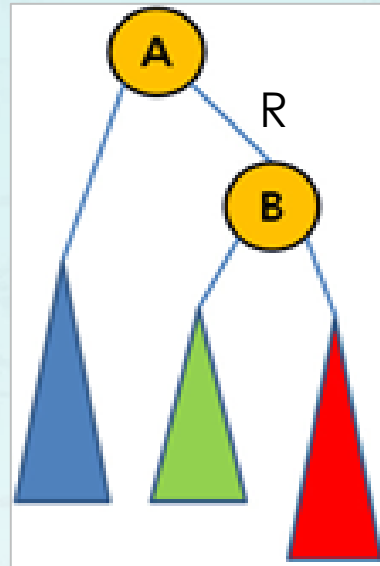
outside case


```
node rotateLL(node A)
{
    node B    = A->left;
    A->left  = B->right;
    B->right = A;
    return B;
}
```

Single Rotation – RR case

outside case


```
node rotateRR(node A)
{
    ?
    ?
    
    return    ?
}
```
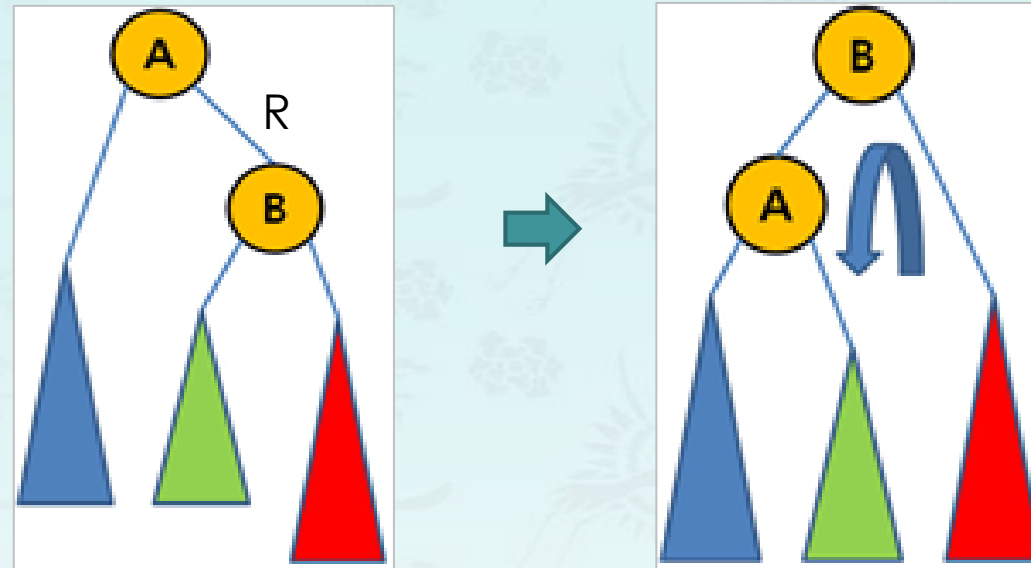
Single Rotation – RR case
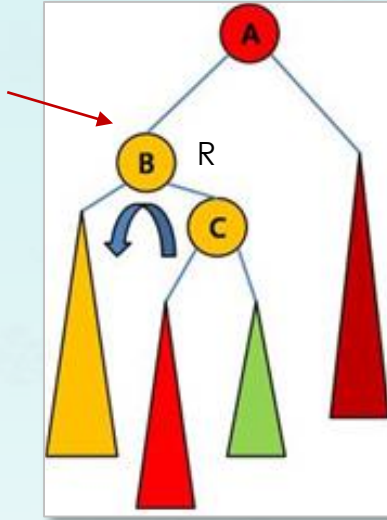
outside case


```
node rotateRR(node A)
{
    node B    = A->right;
    A->right = B->left;
    B->left  = A;
    return B;
}
```
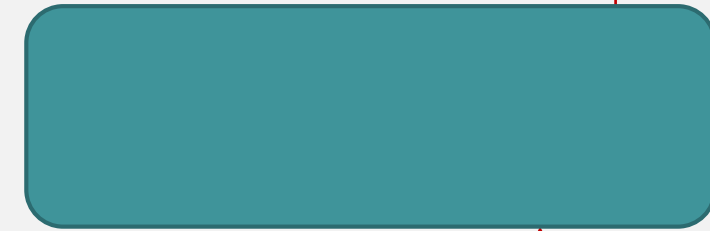
Double Rotation - LR

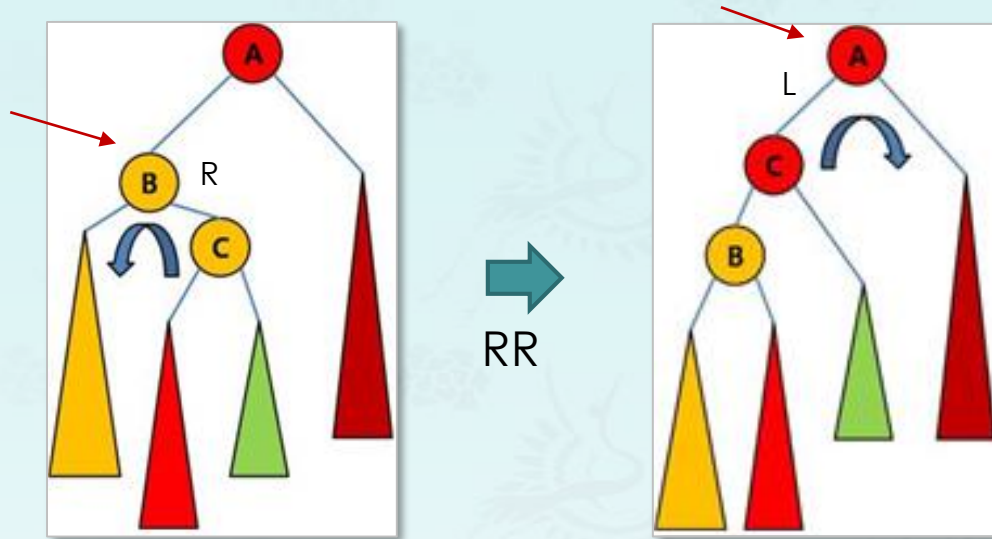inside case


```
node rotateLR(node A) // RR and LL
{



}
```

Two rotations?
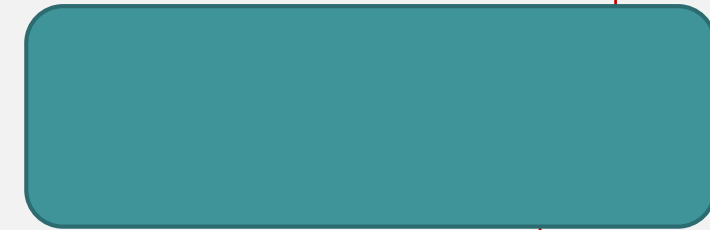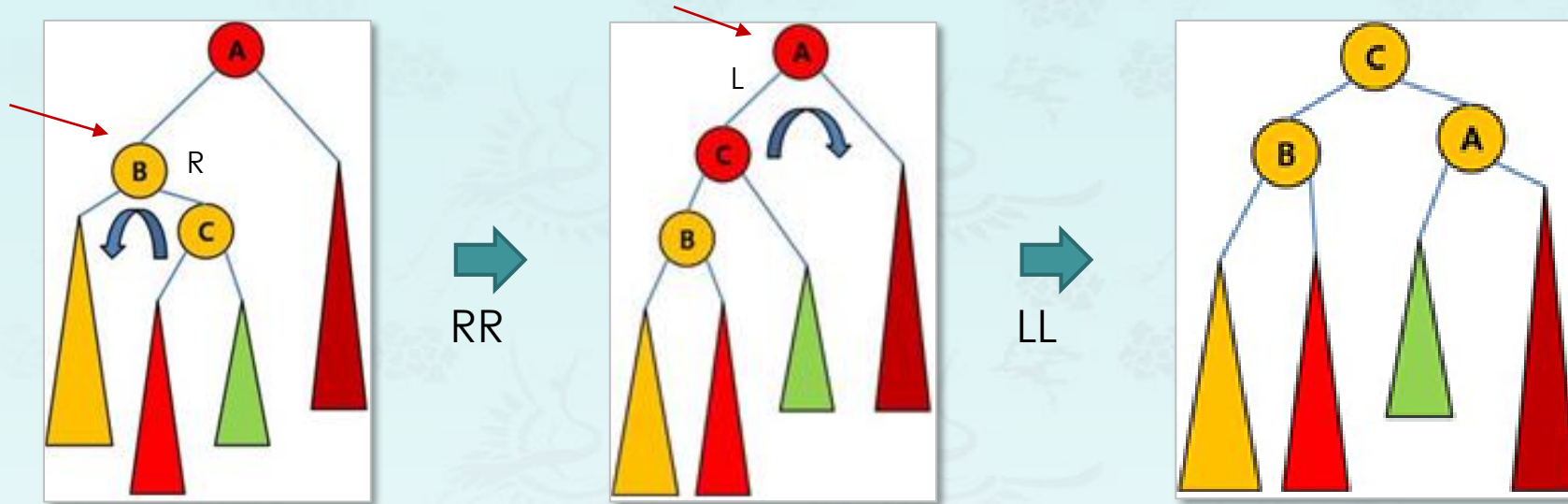
Double Rotation - LR

inside case

RR

```
node rotateLR(node A) // RR and LL
{



}
```

Two rotations?

Double Rotation - LR

inside case

RR

LL

```
node rotateLR(node A) // RR and LL
{


}
```

Two rotations?

Double Rotation - LR

inside case

RR

LL
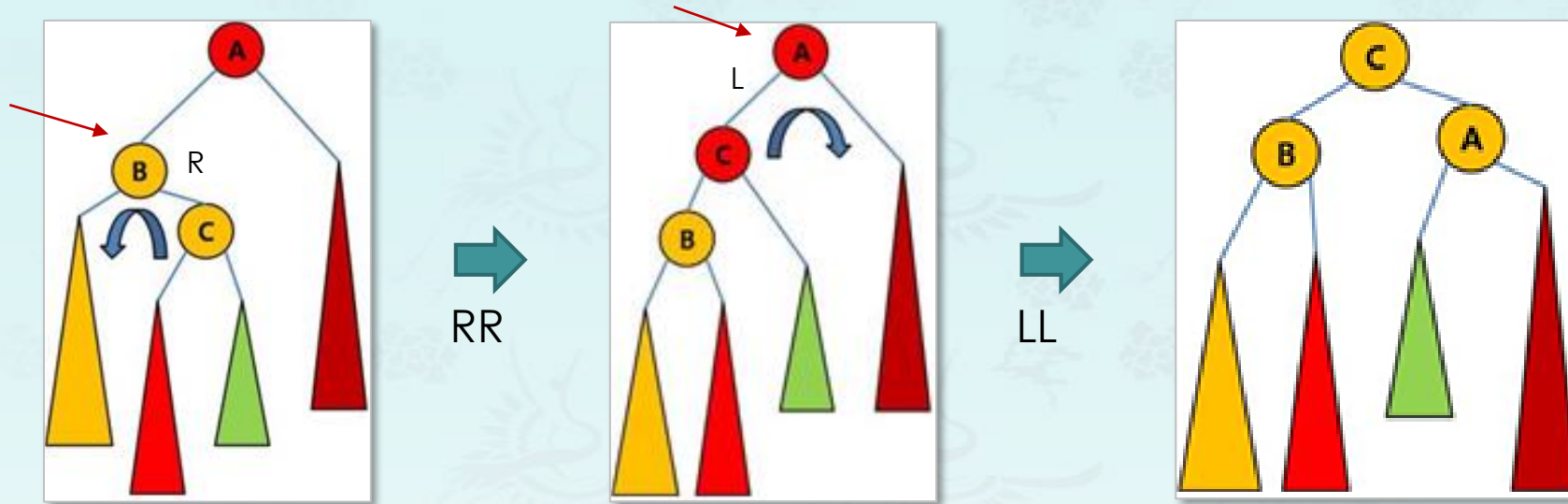
```
node rotateLR(node A) // RR and LL
{
    node B  = A->left;
    A->left = rotateRR(B);
    return rotateLL(A);
}
```

What will return eventually?

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

81

Double Rotation - RL

inside case


```
node rotateRL(node A) { // LL and RR

{



}
```

Two rotations?

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

82
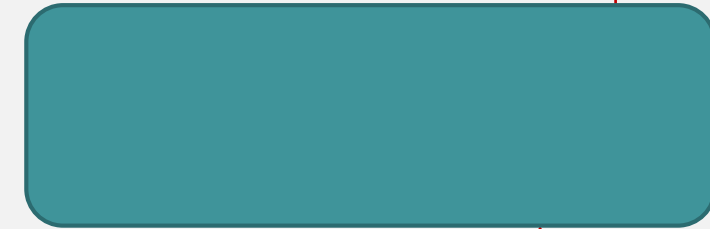
Double Rotation - RL

inside case

LL
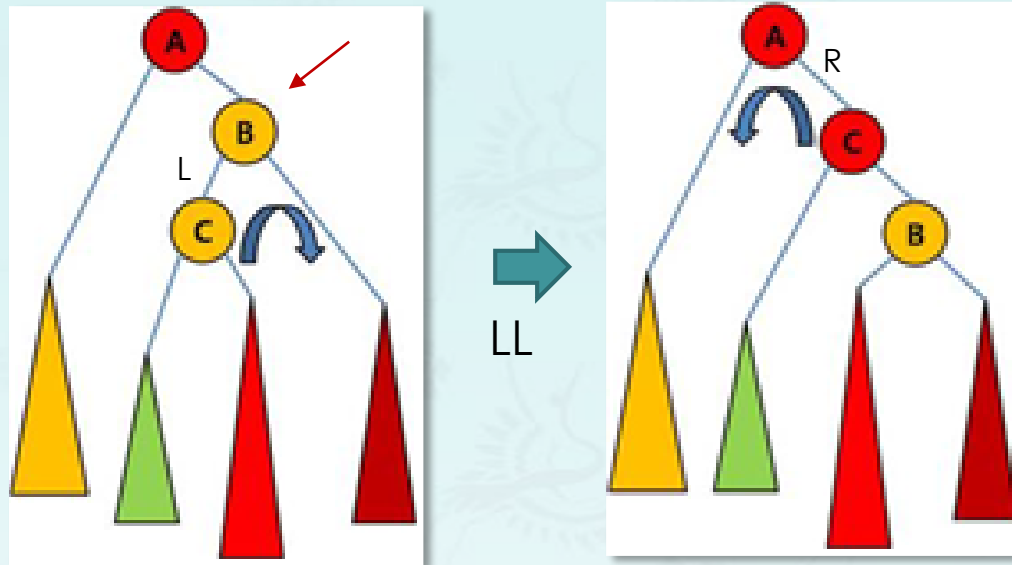
```
node rotateRL(node A) { // LL and RR
{



}
```
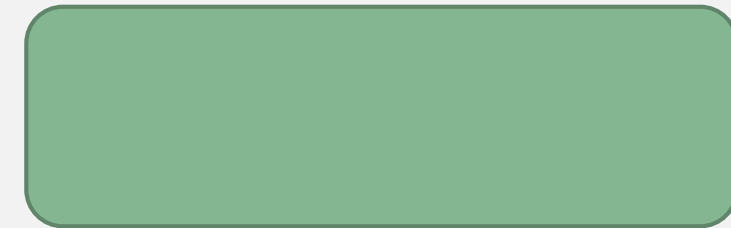
Two rotations?

Double Rotation - RL

inside case

LL

RR

Two rotations?

```
node rotateRL(node A) { // LL and RR
{



}
```
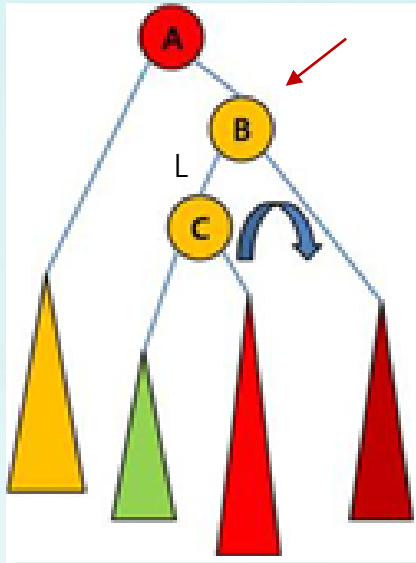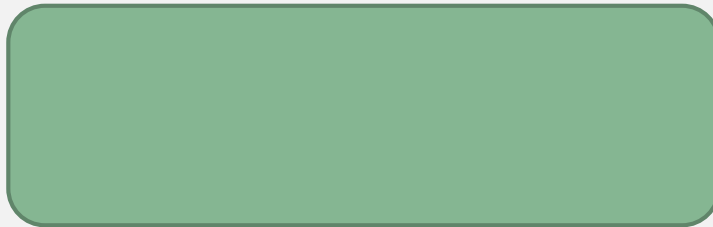
Double Rotation - RL

inside case

LL   RR
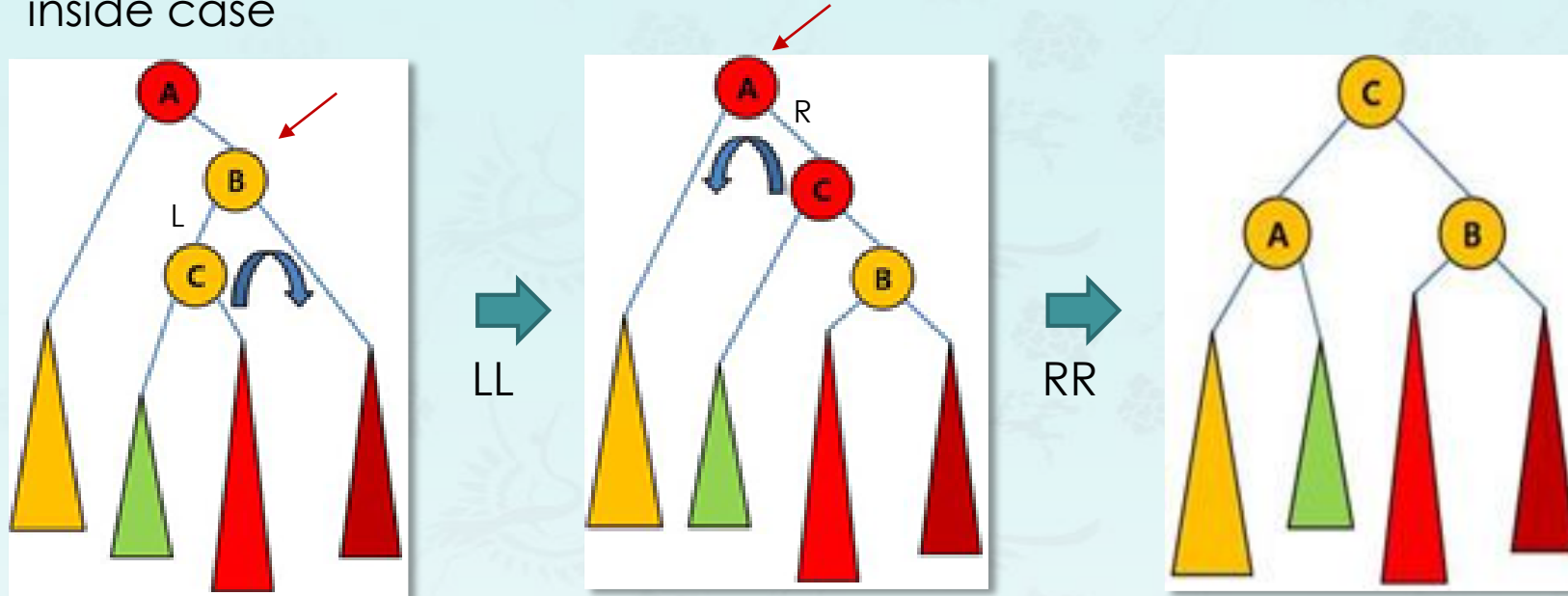
```
node rotateRL(node A) {  // LL and RR
{
    node B  = A->right;
    A->right = rotateLL(B);
    return rotateRR(A);
}
```

Two rotations?
`rotateLL(B)`
`rotateRR(A)`

Insertion of 34
Imbalance at 30

Balance factor at 30 = -2

Double rotation RL
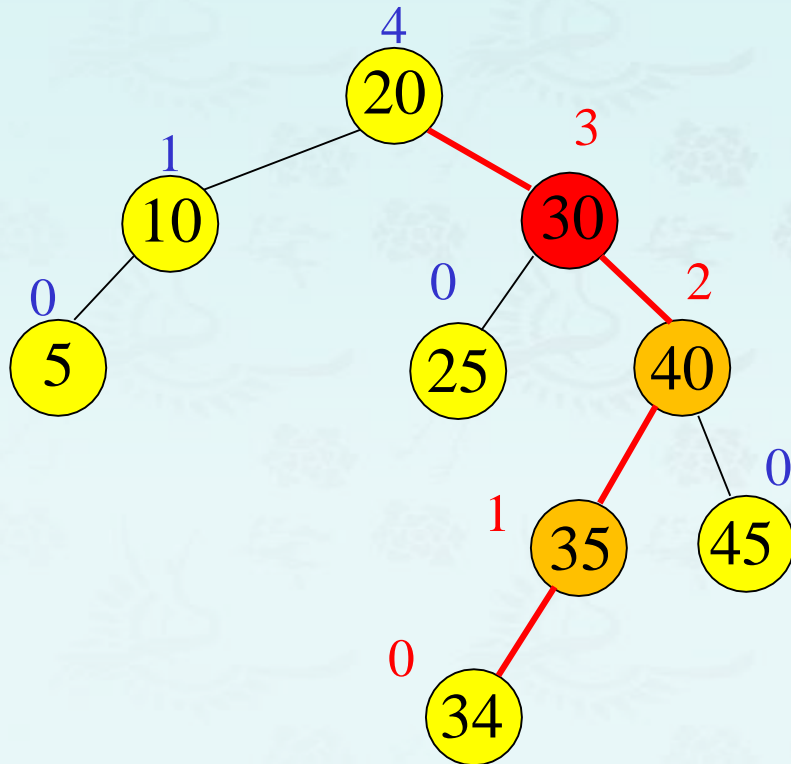
Balance Factor and Height

```
int getHeight(tree node) {
  if (node == NULL) return 0;

  int left  = getHeight (node->left);
  int right = getHeight(node->right);
  return (left > right) ? left + 1 : right + 1;

}
```

```
int balanceFactor(tree node) {
  if (node == NULL) return 0;

  int left  = getHeight(node->left);
  int right = getHeight(node->right);
  return left - right;

}
```

Rebalance

outside case

node

node→left

inside case

node

node→left

outside case

node

node→right

inside case

node

node→right

```
node rebalance(tree node) {
    bf = balanceFactor(node);
    if (bf >= 2) {
        if (balanceFactor(node->left) >= 1)
            node = rotateLL(node);      // LL
                                        outside case
        else
            node = rotateLR(node);      // LR
                                        inside case
    }
    else if (bf <= -2) {
        if (balanceFactor(node->right) <= -1)
            node = rotateRR(node);  outside case
        else
            node = rotateRL(node);  inside case
    }
    return node;
}
```
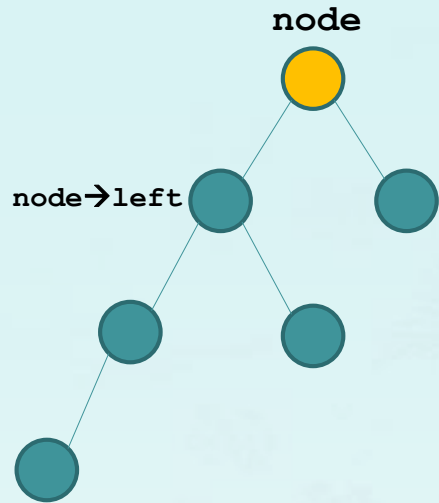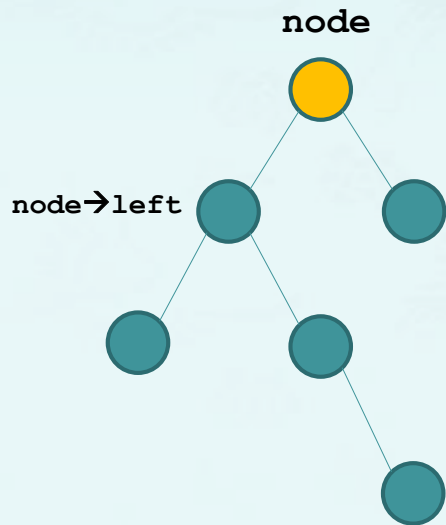
checking single or double rotation

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

96
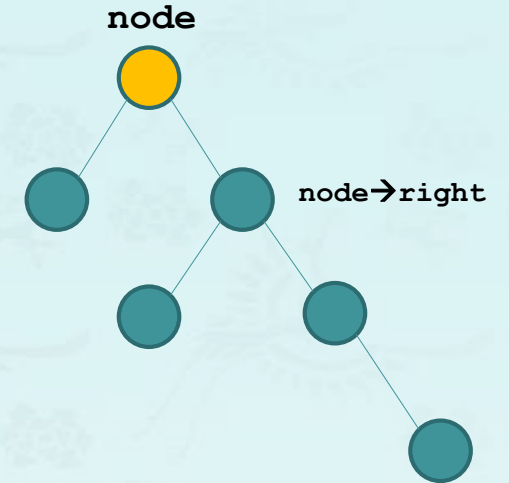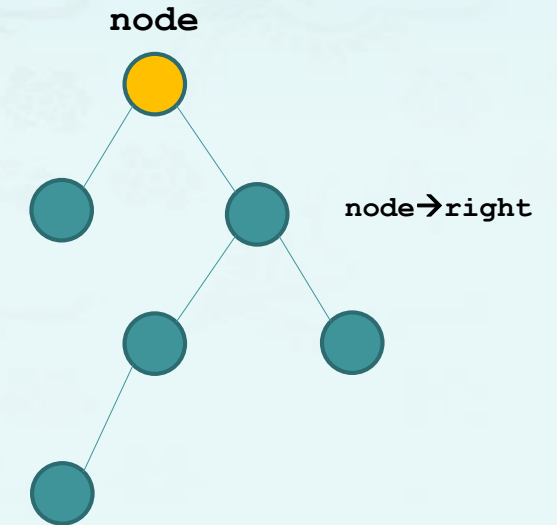
Height of an AVL Tree

N(h) = minimum number of nodes in an AVL tree of height h.

- Basis
  - N(0) = 1, N(1) = 2
- Induction
  - N(h) = N(h-1) + N(h-2) + 1
- Solution (compare it with Fibonacci analysis)
  - N(h) $\geq \phi^h$ ($\phi \approx$ 1.62)

Prof. Youngsup Kim, idebtor@gmail.com, Data Structures, CSEE Dept., Handong Global University

97
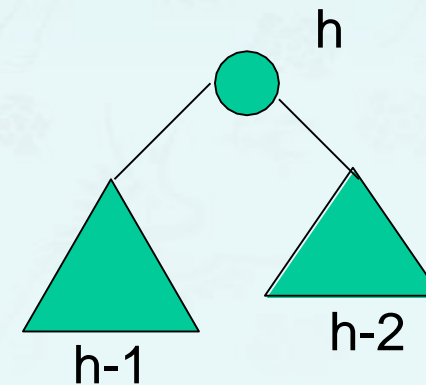
Height of an AVL Tree

- N(h) $\geq \phi^h$ $(\phi \approx 1.62)$

Suppose we have n nodes in an AVL tree of height h.

- $n \geq N(h)$

- $n \geq \phi^h$ hence $log_\phi n \geq h$
  (relatively well balanced tree!!)

- $h \leq 1.44 \, log_2 n$ (i.e., 'Find' operation takes O(log n))

이것도 AVL tree가 될수 있을까요?

저는 AVL tree가 모든 노드의 왼쪽과 오른쪽의 height의 차이가 절대값 1을 넘어서지 않는 것이라고 알고있습니다. 근데 이 트리는 모든 노드에서 왼쪽과 오른쪽의 height의 차이가 절대값 1을 넘지는 않지만 55-54가 연결되어 있는 부분의 높이가 다른 쪽에 비해 2이상 차이나는 것을 보았습니다. 제가 아는 정의상으로는 AVL tree인거 같으면서도 저렇게 height가 2이상 차이가 나니... 결론을 내릴 수가 없어 질문 드립니다.

제가 AVL tree의 정의를 잘못 알고 있는건가요?
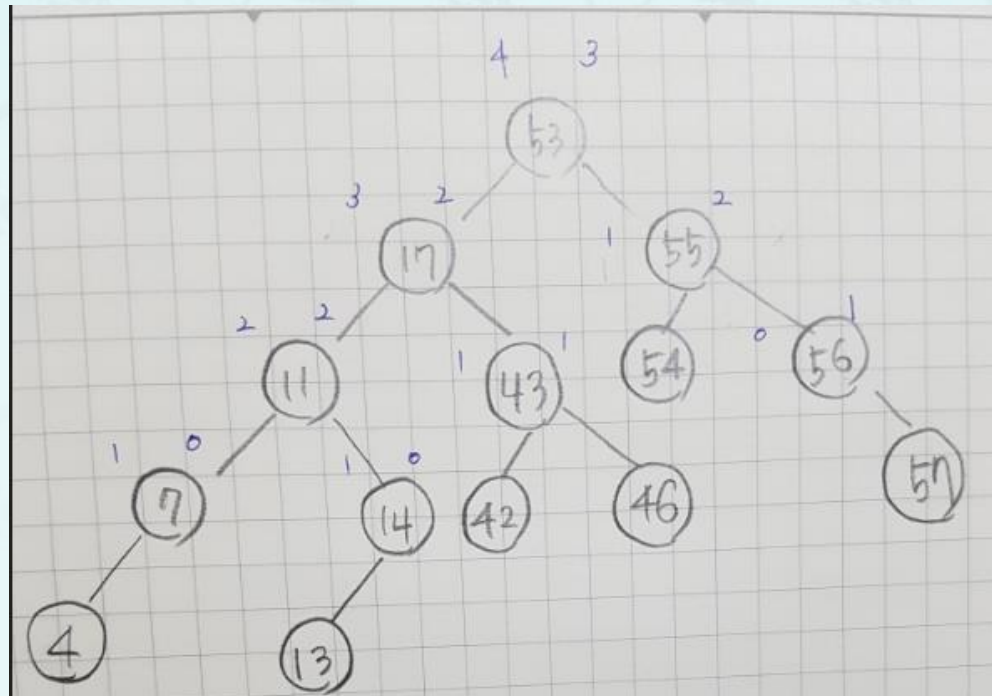
Example with leaf 24 on level 3 and leaf 10 on level 6:

AVL maintain the maximum height difference of 1 between two children subtree, not any two leaves.
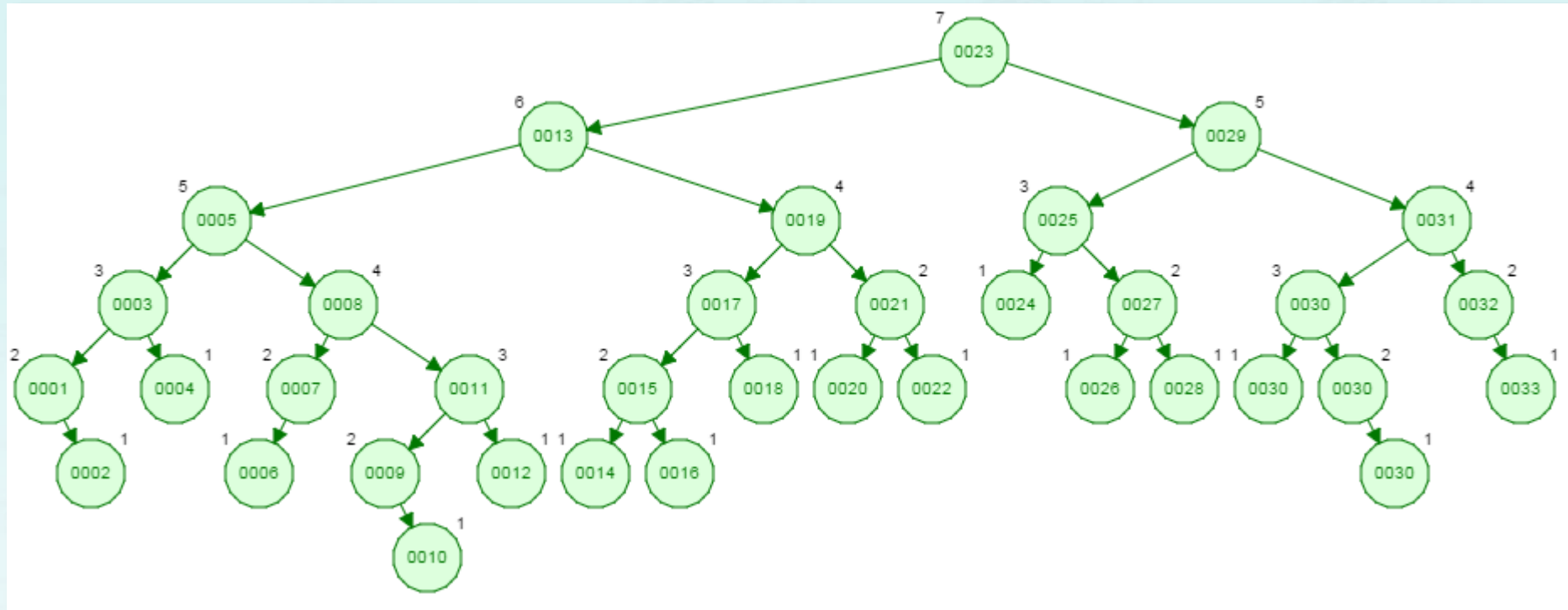
The difference in levels of any two leaves can be any value!
The definition of AVL describes height difference only on two sub-trees from one node.

https://stackoverflow.com/questions/28964971/height-difference-between-leaves-in-an-avl-tree

Pros and Cons of AVL Trees

Arguments **for** AVL trees:

- Search is O(log n) since AVL trees are always balanced.
- Insertion and deletions are also O(log n)
- The height balancing adds no more than a constant factor to the speed of insertion.

Arguments **against** using AVL trees:

- **Difficult** to program & debug; more space for balance factor.
- Asymptotically faster but rebalancing costs time.
- Most large searches are done in database systems on disk and use other structures (e.g. **B-trees**).
- May be OK to have O(N) for a single operation if total run time for many consecutive operations is fast (e.g. Splay trees).

Homework: Draw AVL trees whenever the tree changes its shape by insertion and deletion. Include trees before and after its rotation and the type of rotation.

(1) [1.0p] Insert the sequence of elements (10, 20, 15, 25, 30, 16, 18, 19) into an AVL tree. Delete 30 in the AVL tree that you got above and rebalance it.

(2) [0.5p] Delete 32 in the AVL tree shown below and rebalance it.

▪ Tree가 모양을 바꿀 때마다 AVL tree들을 그려서 다음 시간에 제출합니다.

▪ 각 단계별로 LL, RR, LR, RL을 분명히 표시하십시오. A4 한 두 장 혹은 앞뒤 면을 사용하면 한 장으로도 가능함