AVL Tree Data Structure

261217 Data Structures for Computer Engineers

Patiwet Wuttisarnwattana, Ph.D.

patiwet@eng.cmu.ac.th

Computer Engineering, Chiang Mai University

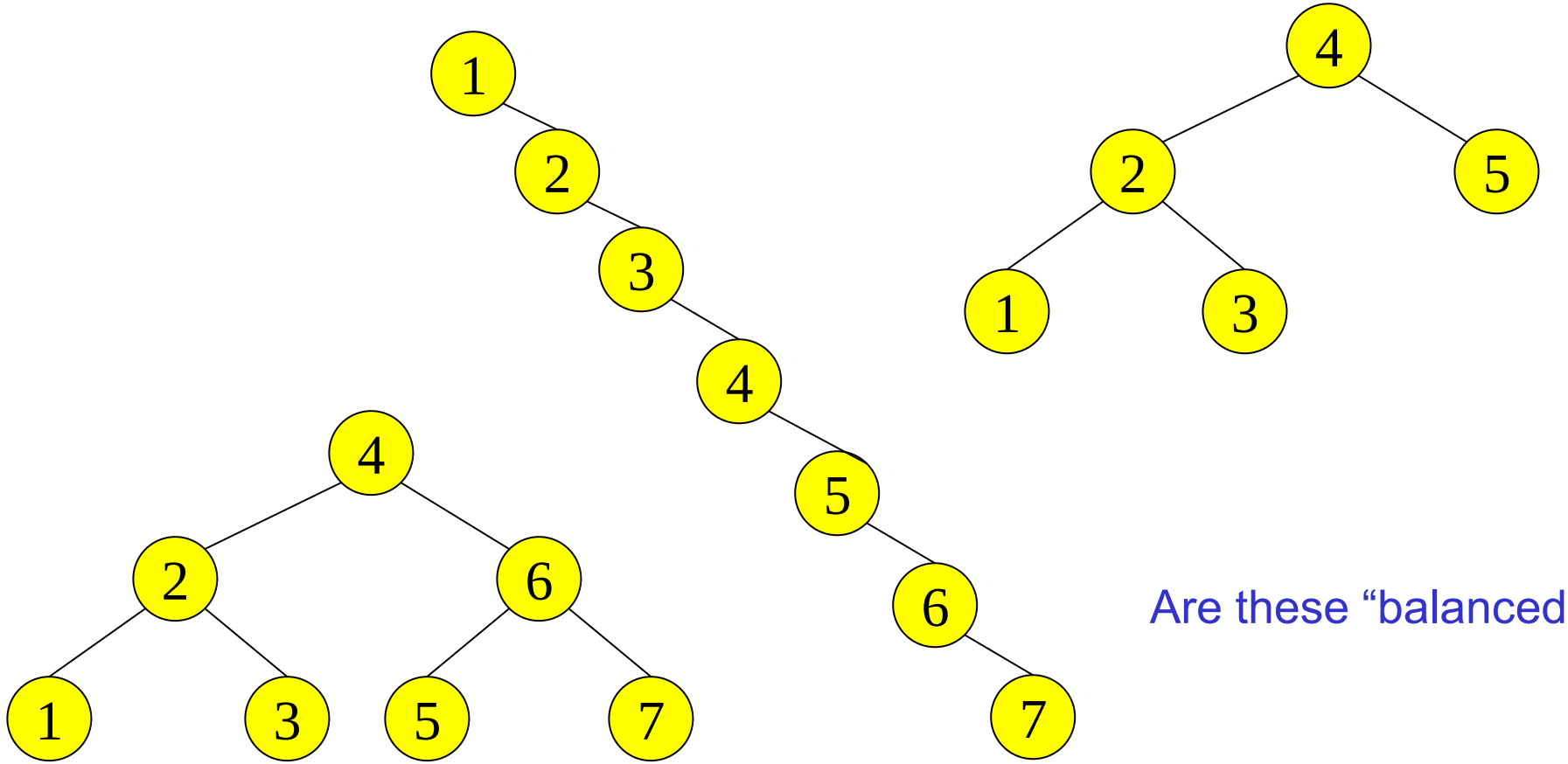
Binary Search Tree: Best Time

- 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)

Binary Search Tree: Worst Time

- 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

Balanced and unbalanced BST



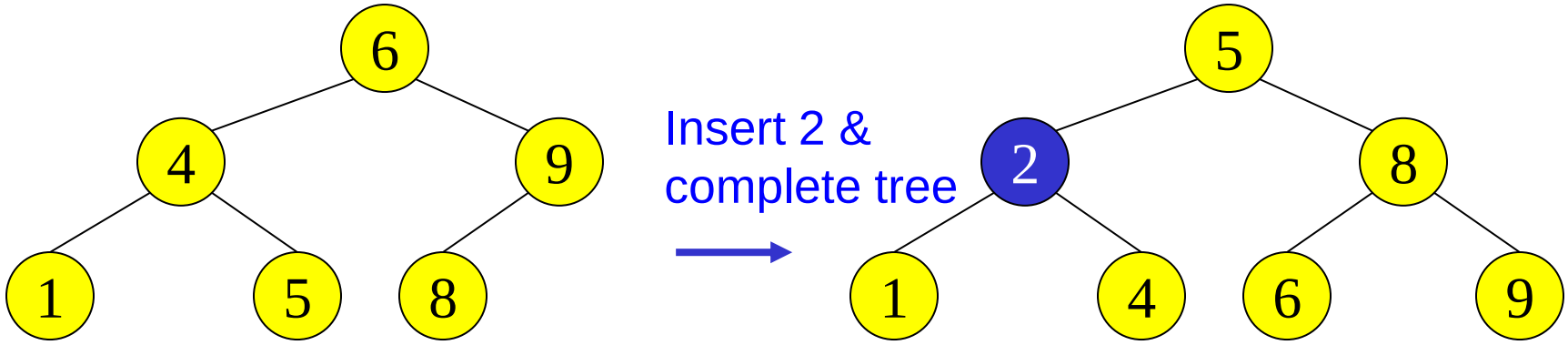
Are these “balanced”?

Approaches to balancing trees

- Don't balance
  - › May end up with some nodes very deep
- Perfect balance
  - › The tree must always be balanced perfectly
- Pretty good balance
  - › Only allow a little out of balance
- Adjust on access
  - › Self-adjusting

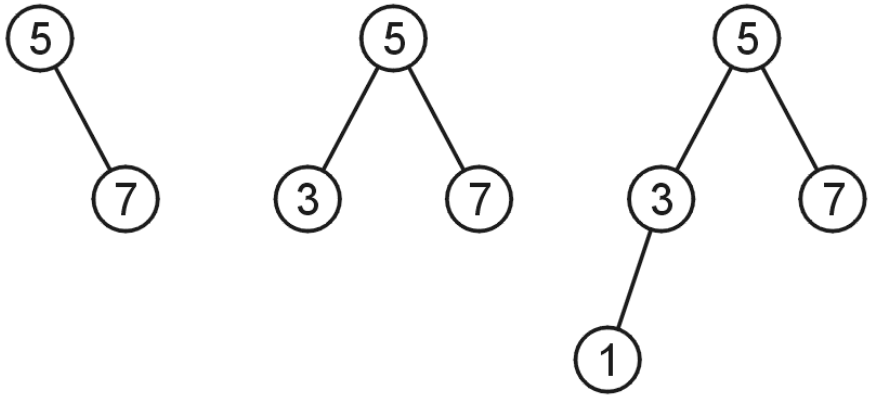
Perfect Balance

- 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



AVL Trees

- Named after Adelson-Velskii and Landis
- AVL trees are height-balanced binary search trees
- Balance factor of a node
  - height(left subtree) - height(right subtree)
  - Balance factor < 0 : “Right-heavy”
  - Balance factor > 0 : “Left-heavy”
  - Balance factor = 0 : “Perfectly balanced”



AVL Trees

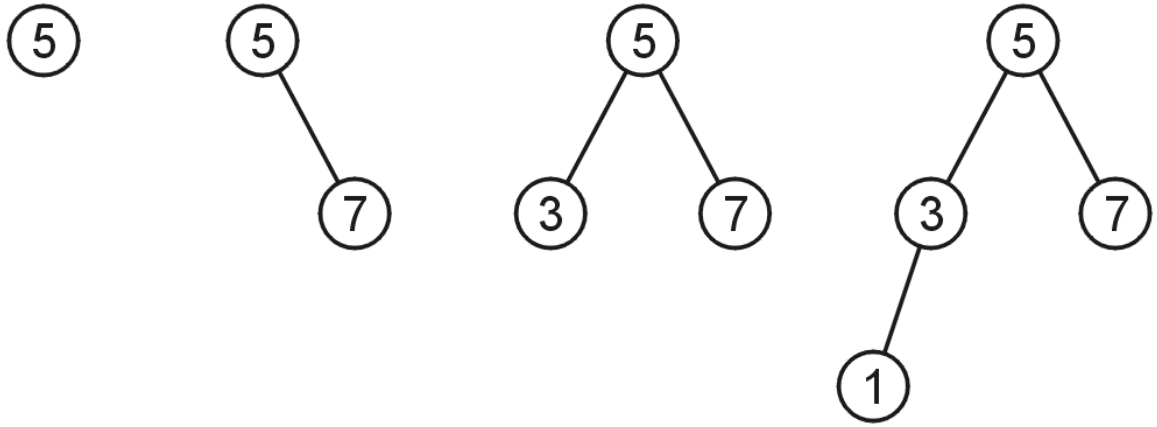
- A binary search tree is a AVL tree if:
  - The difference in the heights between the left and right sub-trees is at most 1, and
  - Both sub-trees are themselves AVL trees

  - |balance factor| ≤ 1
  - |height(left subtree) – height(right subtree)| ≤ 1

  - Store current heights in each node

AVL Trees

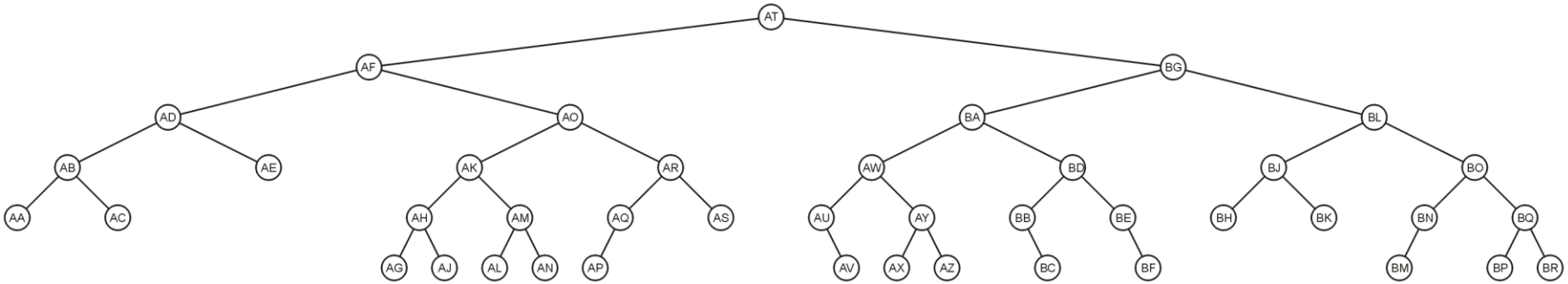
AVL trees with 1, 2, 3, and 4 nodes:



$$|\,\text{height(left tree)} - \text{height(right tree)}\,| \leq 1$$

AVL Trees

Here is a larger AVL tree (42 nodes):

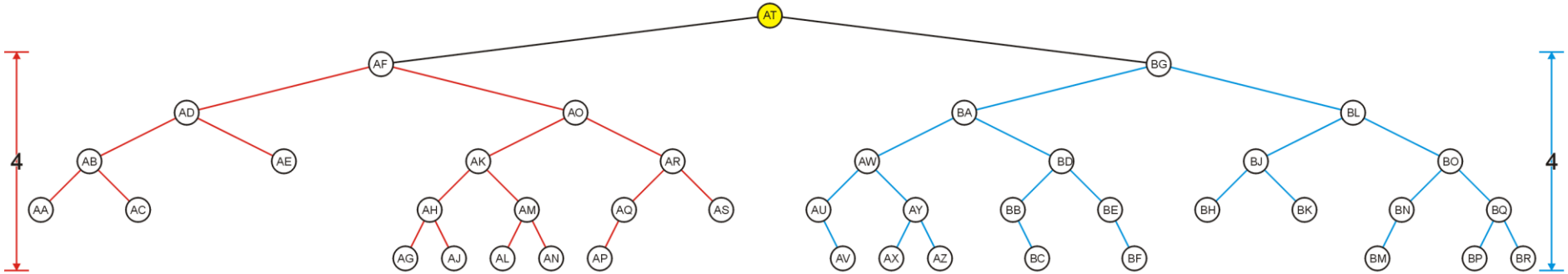


$$|\,\text{height(left tree)} - \text{height(right tree)}\,| \leq 1$$

AVL Trees

The root node is AVL-balanced:

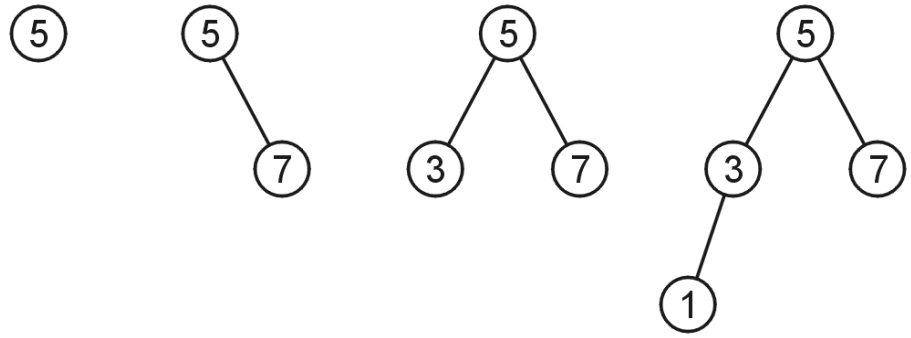
– Both sub-trees are of height 4:



$$|\,\text{height(left tree)} - \text{height(right tree)}\,| \leq 1$$

Height of an AVL Tree

- Let F(h) be the minimum number of nodes in an AVL Tree of height h
  - F(0) = ?
    - 1
  - F(1) = ?
    - 2
  - F(2) = ?
    - 4
  - Can we find F(h)?

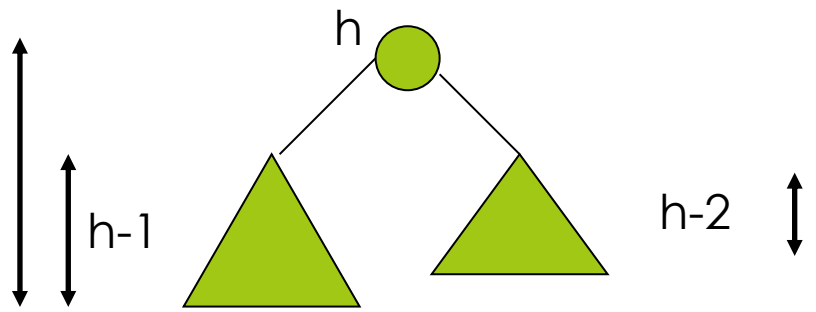


Height of an AVL Tree

The worst-case AVL tree of height h would have:

- A worst-case AVL tree of height $h - 1$ on one side,
- A worst-case AVL tree of height $h - 2$ on the other, and
- The root node

We get: $F(h) = F(h - 1) + 1 + F(h - 2)$



Height of an AVL Tree

This is a recurrence relation:

$$\mathrm{F}(h) = \begin{cases} 1 & h = 0 \\ 2 & h = 1 \\ \mathrm{F}(h-1) + \mathrm{F}(h-2) + 1 & h > 1 \end{cases}$$

The solution?

– Note that $F(h) + 1$ is a Fibonacci number:

$$\begin{aligned} F(0) + 1 &= 2 \rightarrow F(0) = 1 \\ F(1) + 1 &= 3 \rightarrow F(1) = 2 \\ F(2) + 1 &= 5 \rightarrow F(2) = 4 \\ F(3) + 1 &= 8 \rightarrow F(3) = 7 \\ F(4) + 1 &= 13 \rightarrow F(4) = 12 \\ F(5) + 1 &= 21 \rightarrow F(5) = 20 \\ F(6) + 1 &= 34 \rightarrow F(6) = 33 \end{aligned}$$

Height of an AVL Tree

Use Maple Language to solve:

```
> rsolve( {F(0) = 1, F(1) = 2,
           F(h) = 1 + F(h - 1) + F(h - 2)}, F(h) );
```

$$\left(\frac{3\sqrt{5}}{10}+\frac{1}{2}\right)\left(\frac{1}{2}+\frac{\sqrt{5}}{2}\right)^{h}+\left(\frac{1}{2}-\frac{3\sqrt{5}}{10}\right)\left(\frac{1}{2}-\frac{\sqrt{5}}{2}\right)^{h}-\frac{2\sqrt{5}\left(-\frac{2}{1-\sqrt{5}}\right)^{h}}{5\left(1-\sqrt{5}\right)}+\frac{2\sqrt{5}\left(-\frac{2}{1+\sqrt{5}}\right)^{h}}{5\left(1+\sqrt{5}\right)}-1$$

```
> asympt( %, h );
```

$$\frac{\left(3\sqrt{5}+5\right)\left(1+\sqrt{5}\right)^{h}}{5\left(-1+\sqrt{5}\right)2^{h}}-1$$

$$c\left(\frac{1+\sqrt{5}}{2}\right)^{h}$$

Height of an AVL Tree

This is approximately

$$F(h) \approx 1.8944\ \phi^h - 1$$

where $\phi \approx 1.6180$ is the golden ratio $\left(\frac{1+\sqrt{5}}{2}\right)$

- That is, $F(h) = \Omega(\phi^h)$

Thus, we may find the maximum value of h for a given n:

$$\log_\phi\left(\frac{n+1}{1.8944}\right) = \log_\phi(n+1) - 1.3277 = 1.4404 \cdot \lg(n+1) - 1.3277$$

Height of an AVL Tree

- Let's N(h) = F(h)
- $N(h) \geq \phi^h \quad (\phi \approx 1.62)$
- Suppose we have n nodes in an AVL tree of height h.
  - $n \geq N(h)$ (because N(h) was the minimum)
  - $n \geq \phi^h$ hence $\log_\phi n \geq h$ (relatively well balanced tree!!)
  - $h \leq 1.44 \log_2 n$ (i.e., Find takes O(log n))

Maintaining Balance

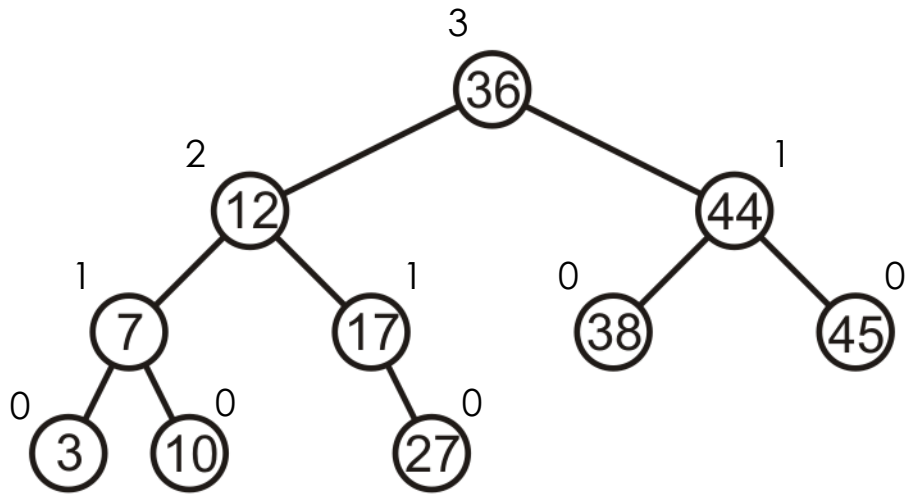
To maintain AVL balance, observe that:

- Inserting a node can increase the height of a tree by at most 1
- Removing a node can decrease the height of a tree by at most 1

Maintaining Balance

Is this an AVL tree?

Inserting 15 into this tree?

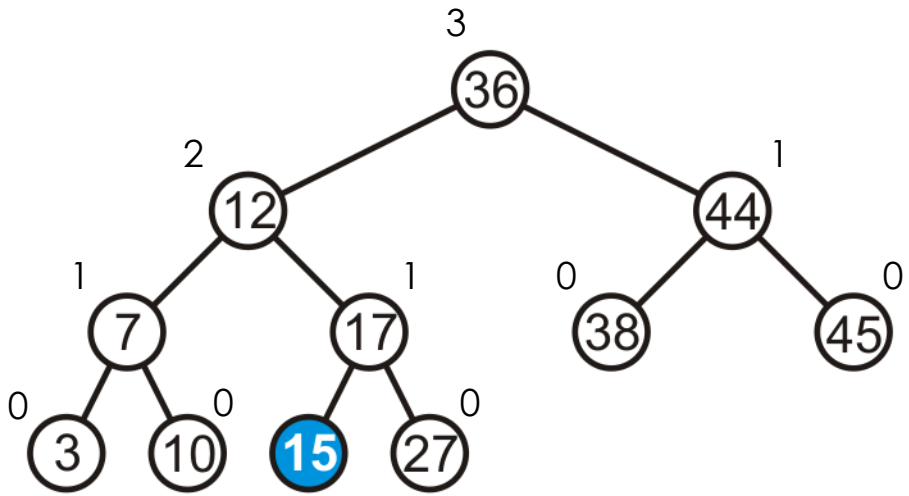


| Height(Left_Subtree) – Height(Right_Subtree) | ≤ 1

Maintaining Balance

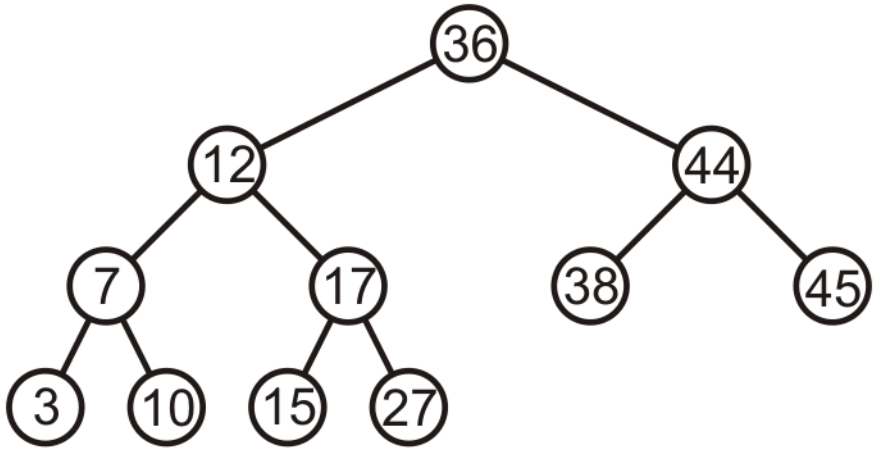
Consider inserting 15 into this tree

– In this case, the heights of none of the trees change



Maintaining Balance

The tree remains balanced

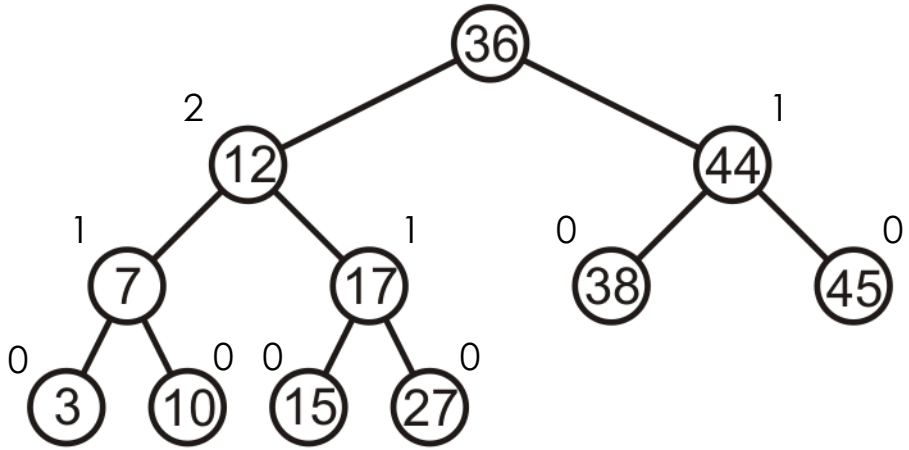


| Height(Left_Subtree) – Height(Right_Subtree) | ≤ 1

Maintaining Balance

The tree remains balanced

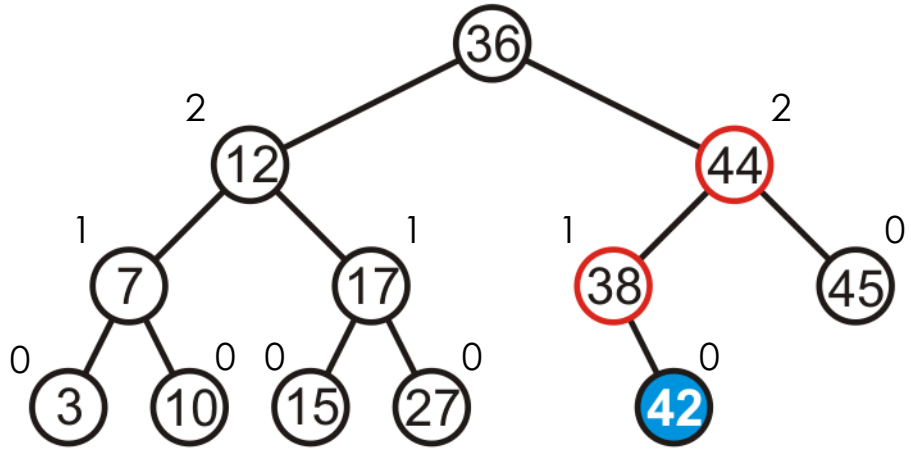
Consider inserting 42 into this tree?



Maintaining Balance

Consider inserting 42 into this tree

- Now we see the heights of two sub-trees have increased by one
- The tree is still balanced



$$|\,\text{Height}(\text{Left\_Subtree}) - \text{Height}(\text{Right\_Subtree})\,| \leq 1$$

Maintaining Balance

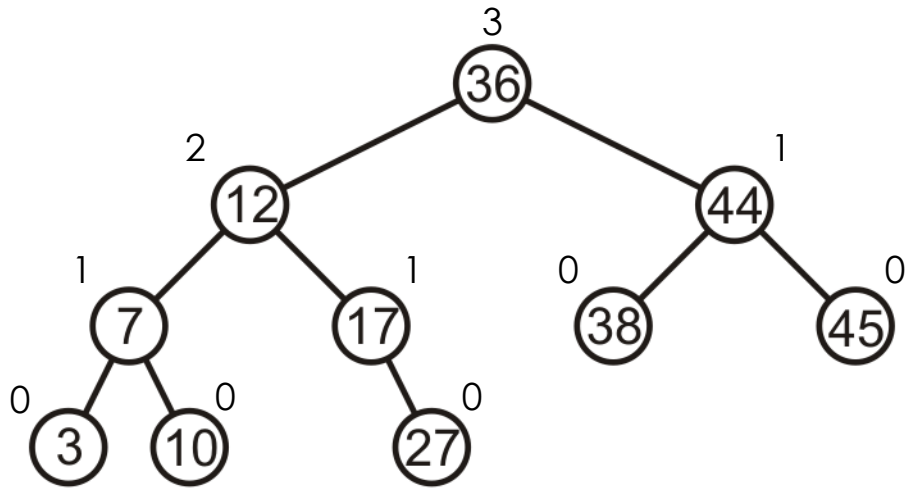
- We should store node height at each node
- Update the variable only when inserting and deleting
- Update the height only along the path from the node to the root
- These algorithms are recursive

Maintaining Balance

If a tree is AVL tree is going to be imbalance due to an insertion:

- The insertion must increase the height of the deeper sub-tree by 1
- |Height(Left subtree) – Height(Right subtree)| > 1

Insert(23)?

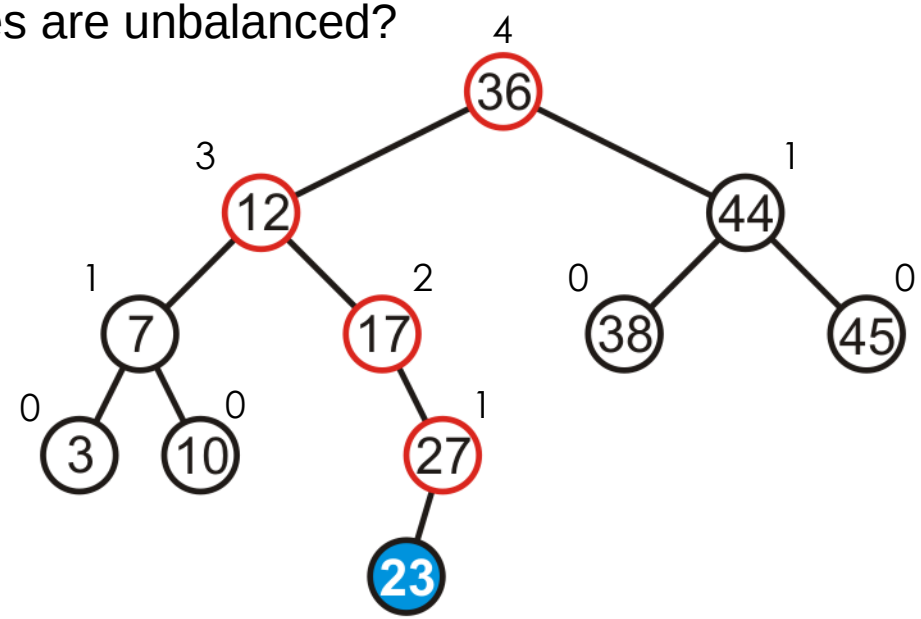


Maintaining Balance

Insert(23)

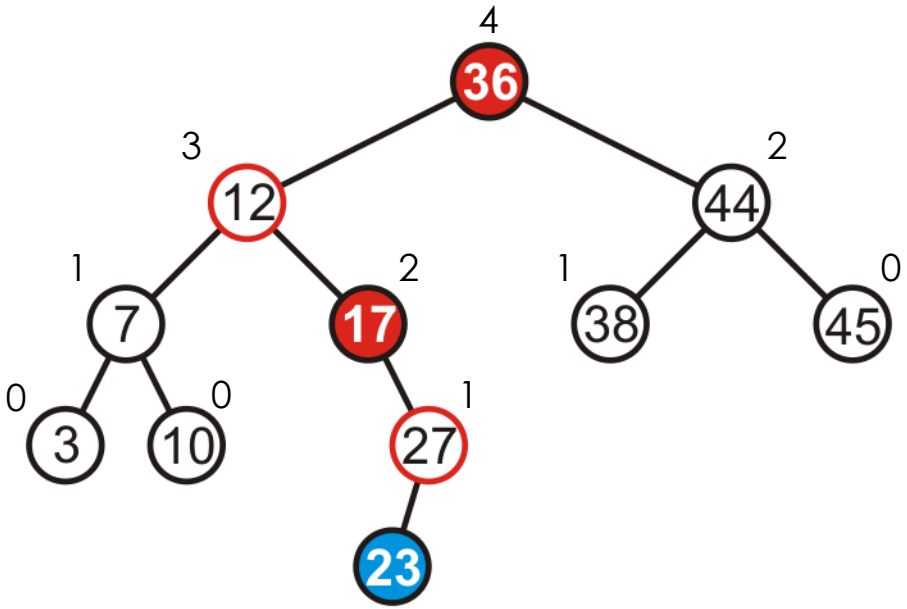
The heights of each of the sub-trees from here to the root are increased by one

Which nodes are unbalanced?



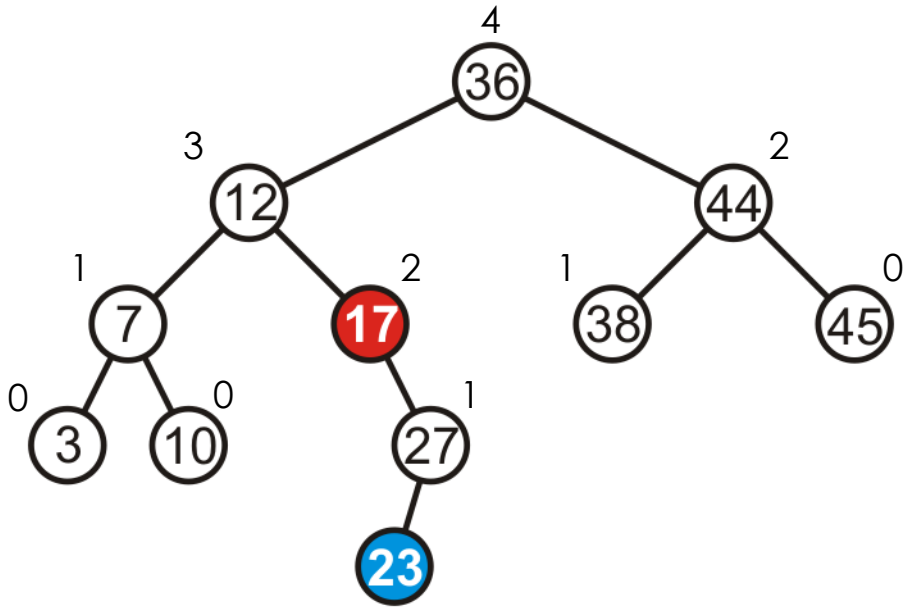
Maintaining Balance

Only nodes 17 and 36 are unbalanced



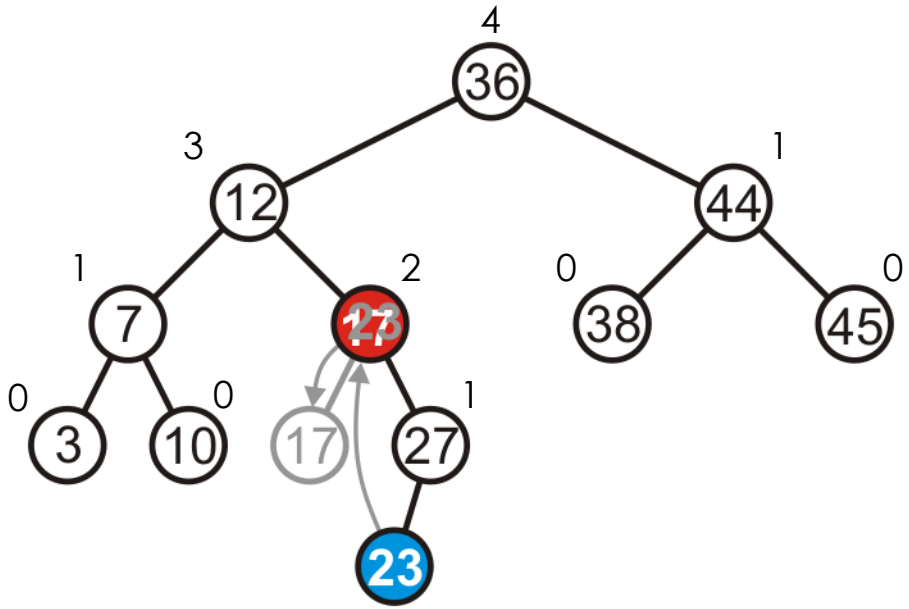
Maintaining Balance

Let's fix the lowest node first (Node 17), any idea???



Maintaining Balance

Let's promote 23 to where 17 is, and make 17 the left child of 23

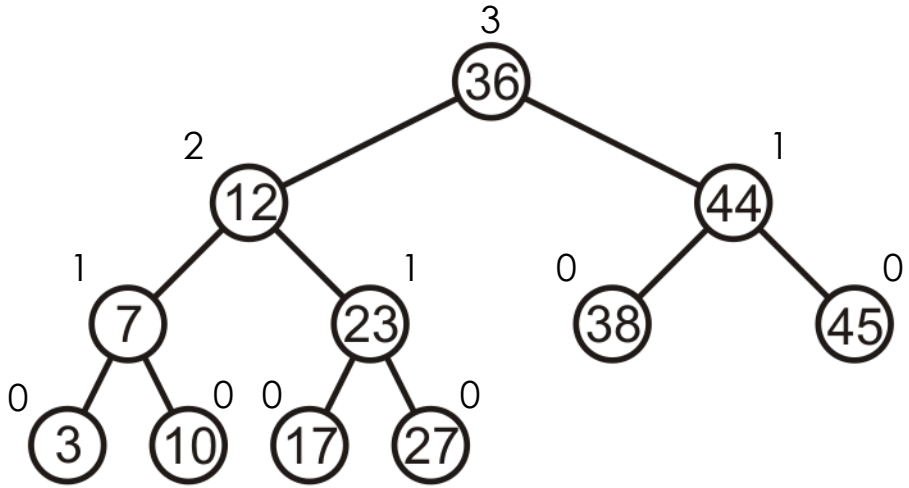


Maintaining Balance

Thus, that node is no longer unbalanced (AVL tree)

- Incidentally, neither is the root now balanced again, too

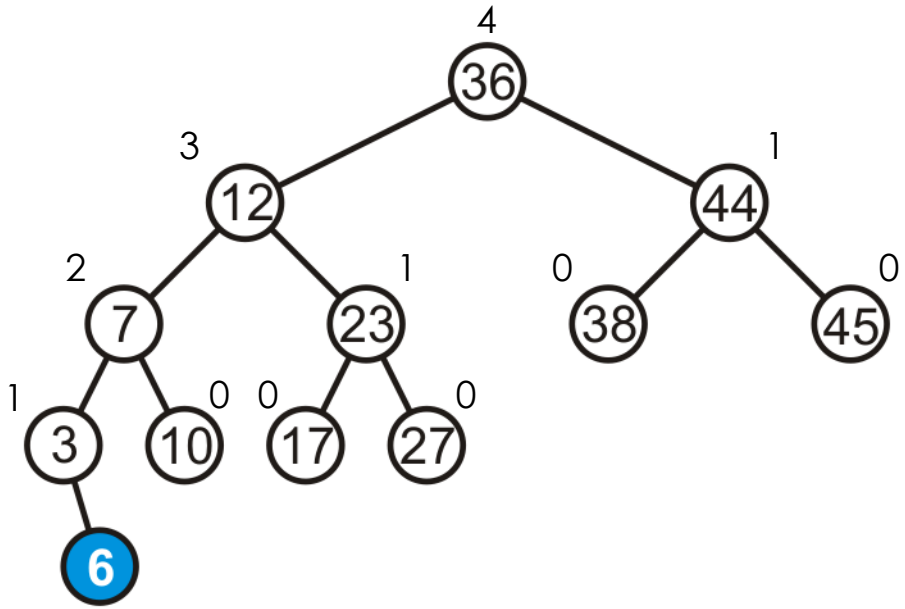
- Now, Insert(6)?



Maintaining Balance

Consider adding 6:

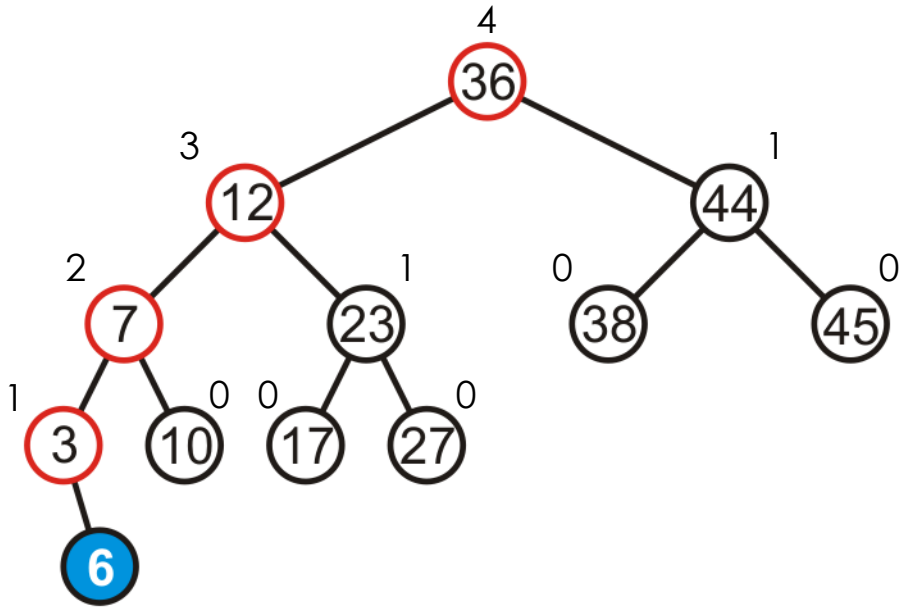
Any node has to be updated its height?



Maintaining Balance

The height of each of the trees in the path back to the root are increased by one

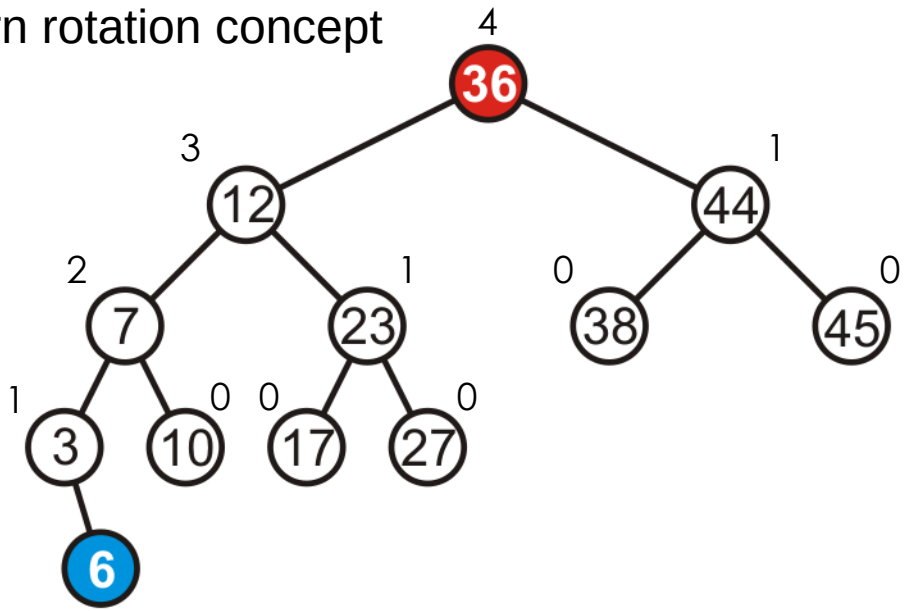
Any node is imbalance?



Maintaining Balance

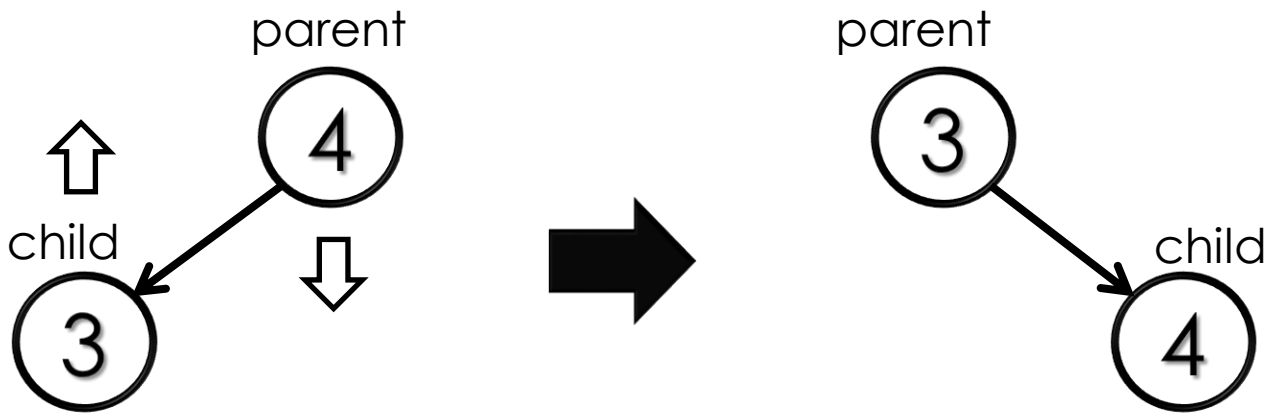
The height of each of the trees in the path back to the root are increased by one

- However, only the root node is now unbalanced
- How to fix this, any idea? (difficult?)
- Need to learn rotation concept



Rotation concept

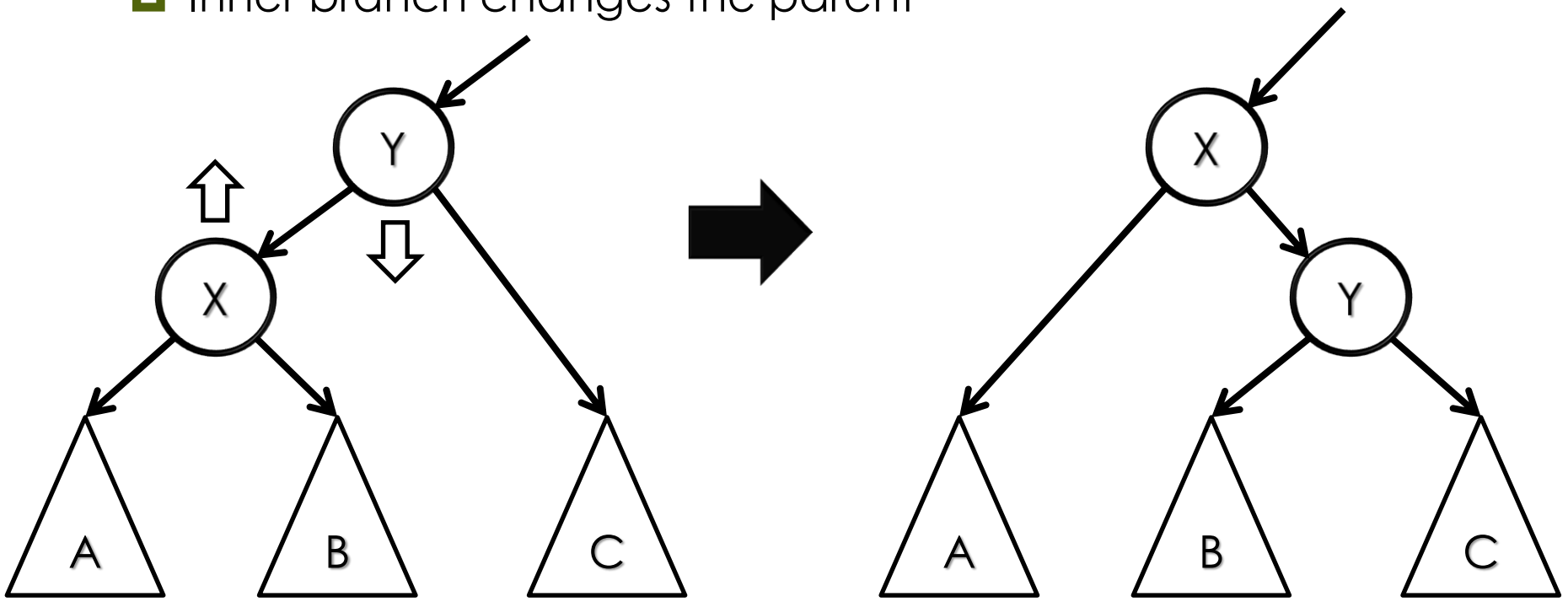
- Rotation is to switch role between two nodes
  - Parent -> Child
  - Child -> Parent
  - After the rotation, the binary search tree property must still hold
    - Left child is smaller than the parent
    - Right child is larger than the parent



These two tree are equivalent

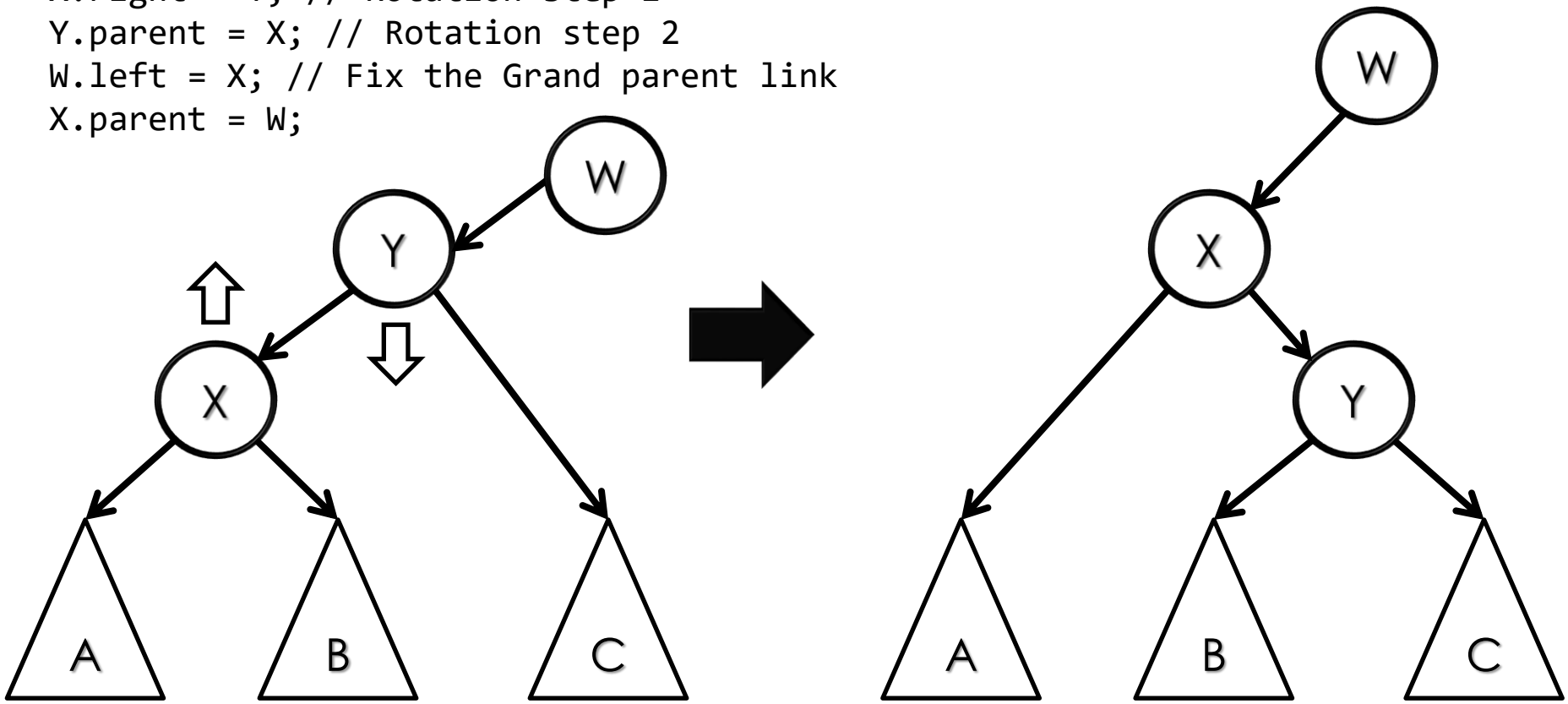
Rotation concept

- Single rotation (Single rotation from left)
  - Switch role between X and Y (child -> parent)
  - Outer branches stay the same
  - Inner branch changes the parent



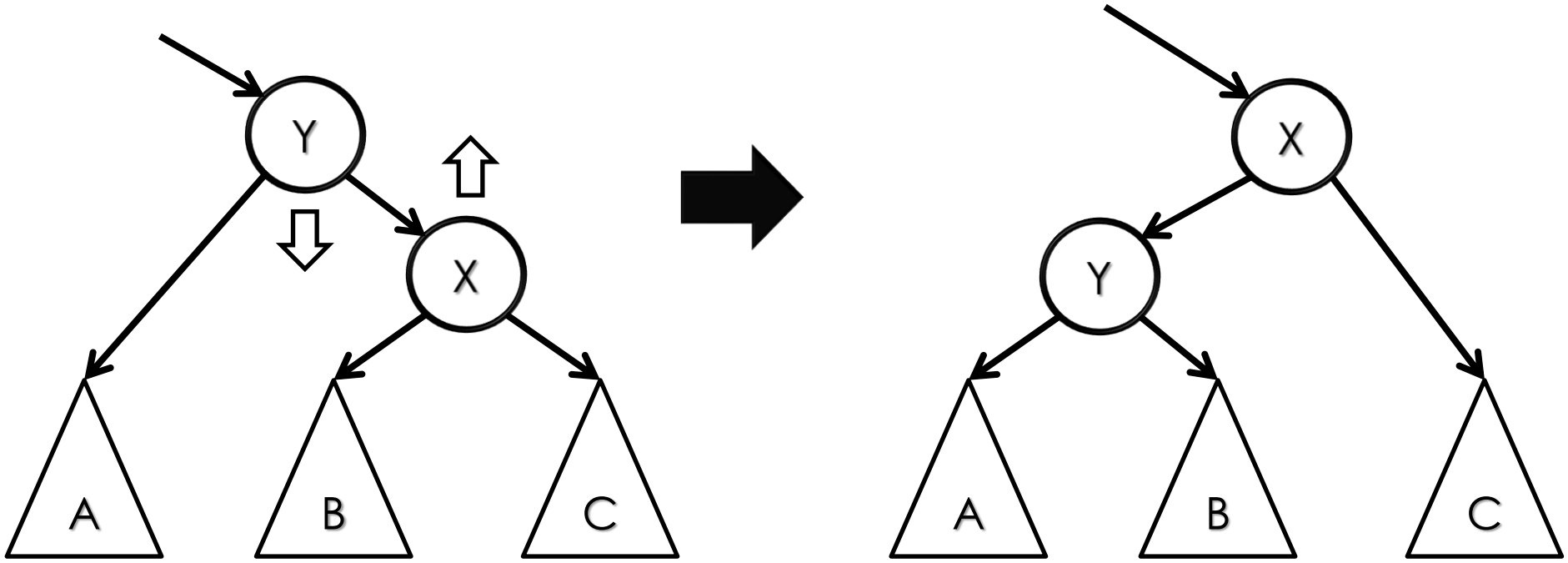
Implementation

```
Node Y = node;
Node X = node.left;
Node W = node.parent;
Y.left = X.right; // Transfer the inner branch
Y.left.parent = Y;
X.right = Y; // Rotation step 1
Y.parent = X; // Rotation step 2
W.left = X; // Fix the Grand parent link
X.parent = W;
```

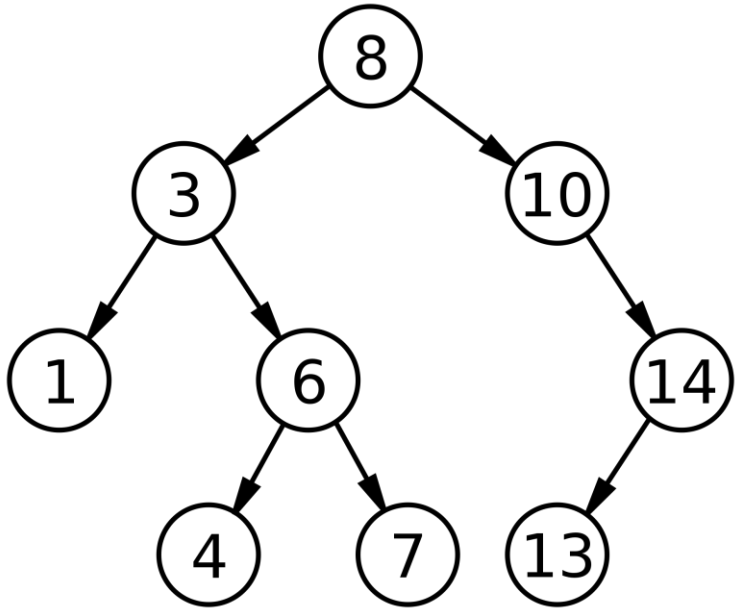


Rotation concept

- Single rotation (Single rotation from right)
  - Switch role between X and Y (child -> parent)
  - Outer branches stay the same
  - Inner branch changes the parent



Rotation concept (Examples)



Show the results of the following operations

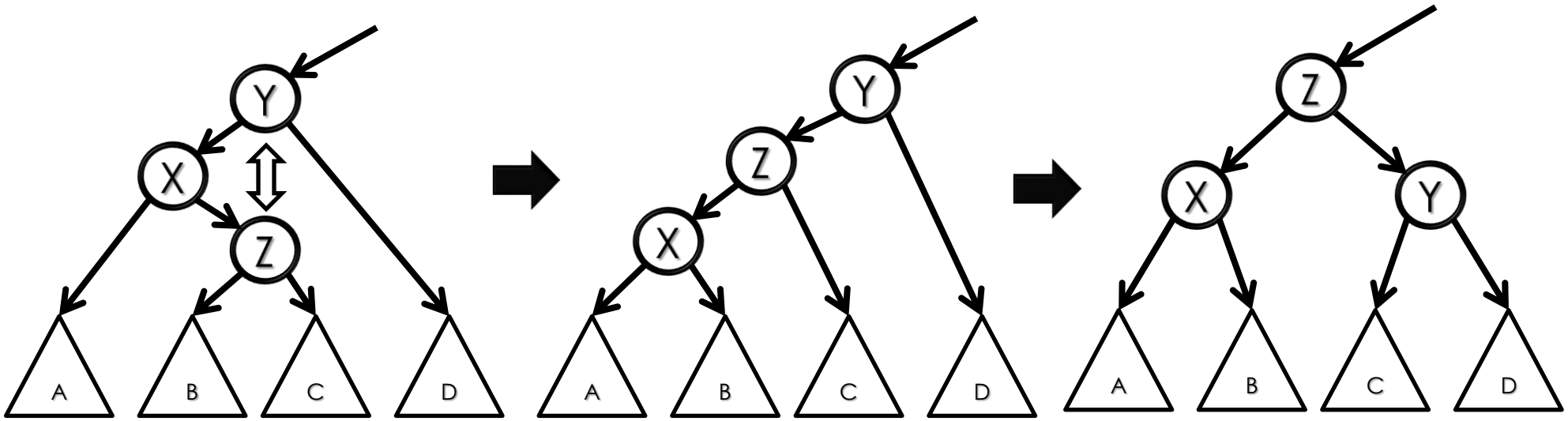
- SingleRotateFromLeft(6)
- SingleRotateFromRight(6)
- SingleRotateFromLeft(3)
- SingleRotateFromRight(3)
- SingleRotateFromLeft(8)
- SingleRotateFromRight(8)

Tricks

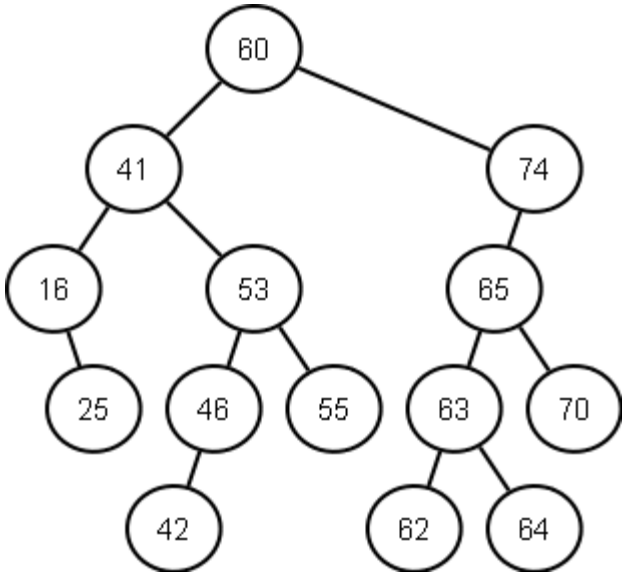
- Outer branches stay the same
- Inner branch is transferred to the other

Rotation concept

- Double rotation (Inner Case)
  - Switch position between Y and Z (grand child <-> grand parent)
  - Rotate X and Z then rotate Z and Y
    - The outer branches stay the same; the inner branch changes the parent



Rotation concept (Examples)



Show the results of the following operations

- DoubleRotateFromInnerLeft(41)
- DoubleRotateFromInnerRight(41)
- DoubleRotateFromInnerLeft(60)
- DoubleRotateFromInnerRight(60)

- DoubleRotateFromOuterLeft(60)

Tricks

- Rotate Z,X then Rotate Z,Y
- Outer branches stay the same
- Inner branch is transferred to the other

Insertion causes AVL Trees to be imbalance

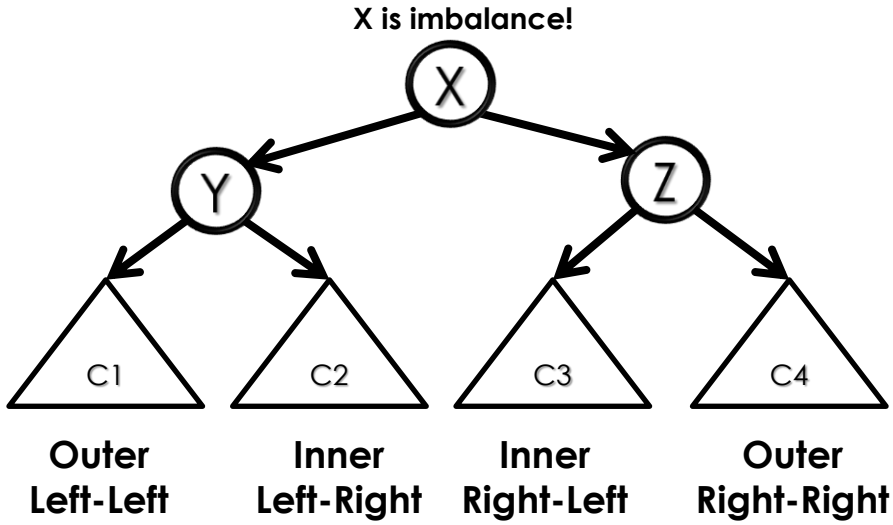
Let the node that needs rebalancing be X.
There are 4 cases:

Outer Cases (require **single rotation**) :

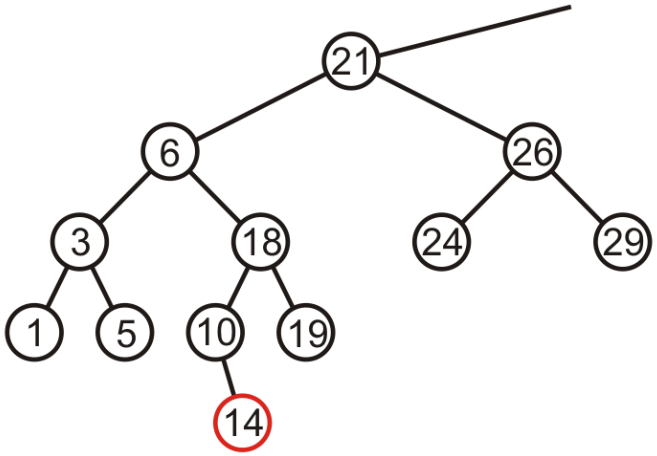
1. Insertion into left subtree of left child of X.
2. Insertion into right subtree of right child of X.

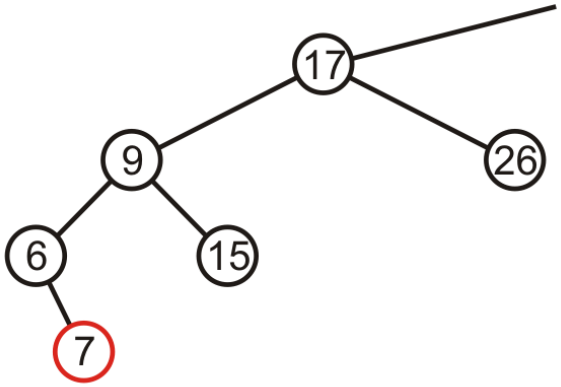
Inner Cases (require **double rotation**) :

3. Insertion into right subtree of left child of X.
4. Insertion into left subtree of right child of X.



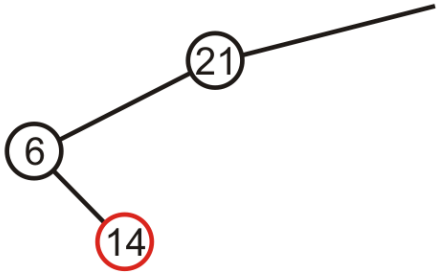
Inner or Outer

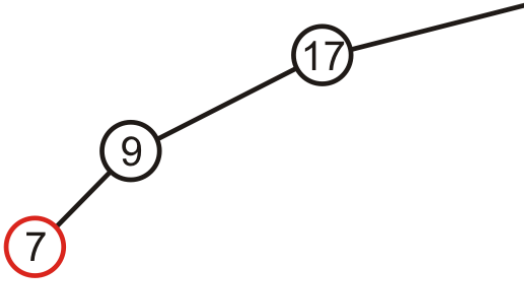


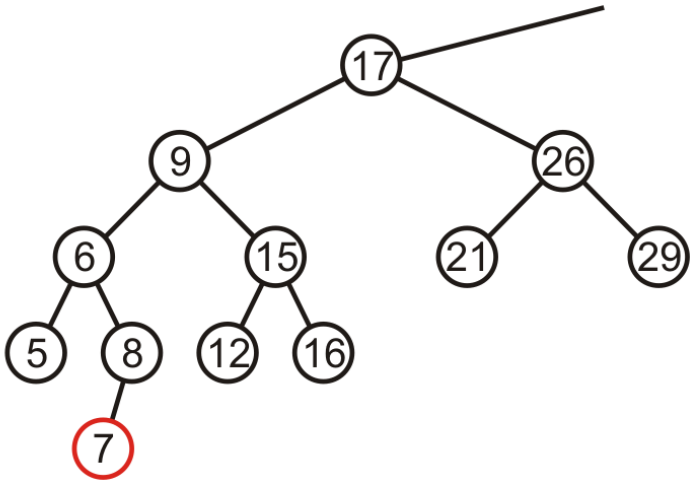


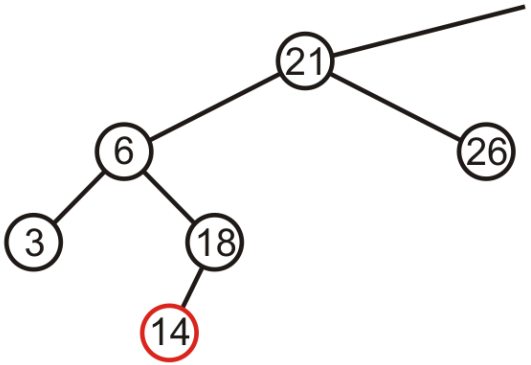
Which one is imbalance?

Look at the root!!!





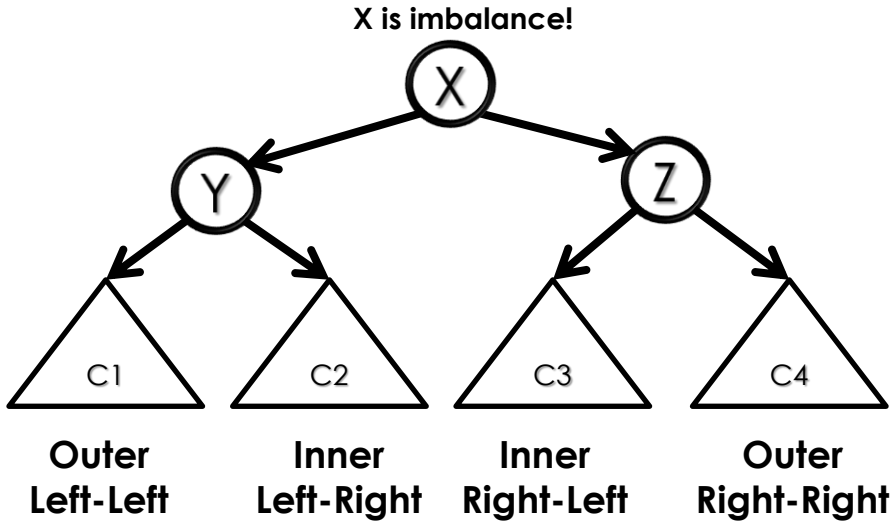




Solution: Rotation from the insertion direction

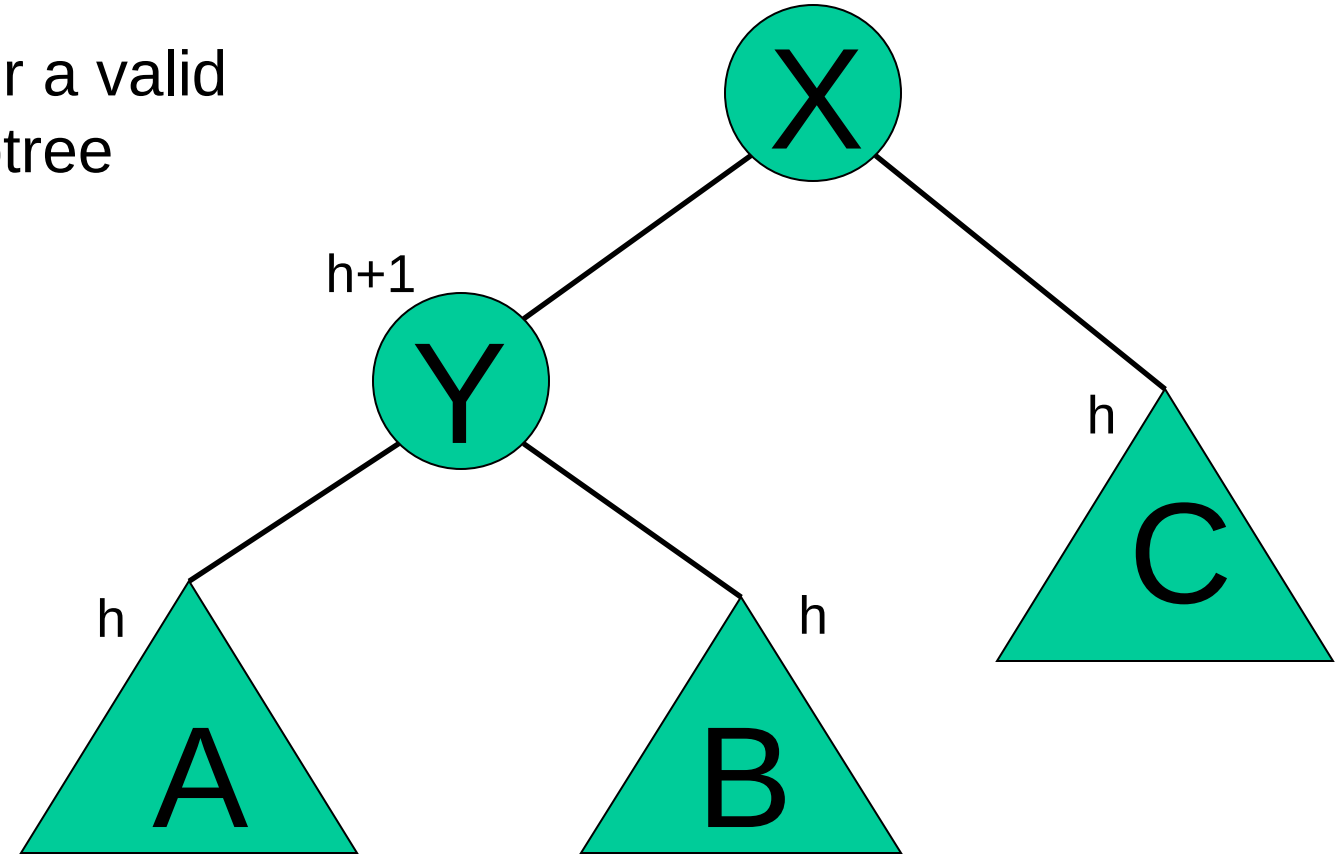
Let the node that needs rebalancing be X.
Solution for each case:

1. If the insert direction is Left-Left (Outer) → SingleRotationFromLeft(X)
2. If the insert direction is Right-Right (Outer) → SingleRotationFromRight(X)
3. If the insert direction is Left-Right (Inner) → DoubleRotationFromLeft(X)
4. If the insert direction is Right-Left (Inner) → DoubleRotationFromRight(X)



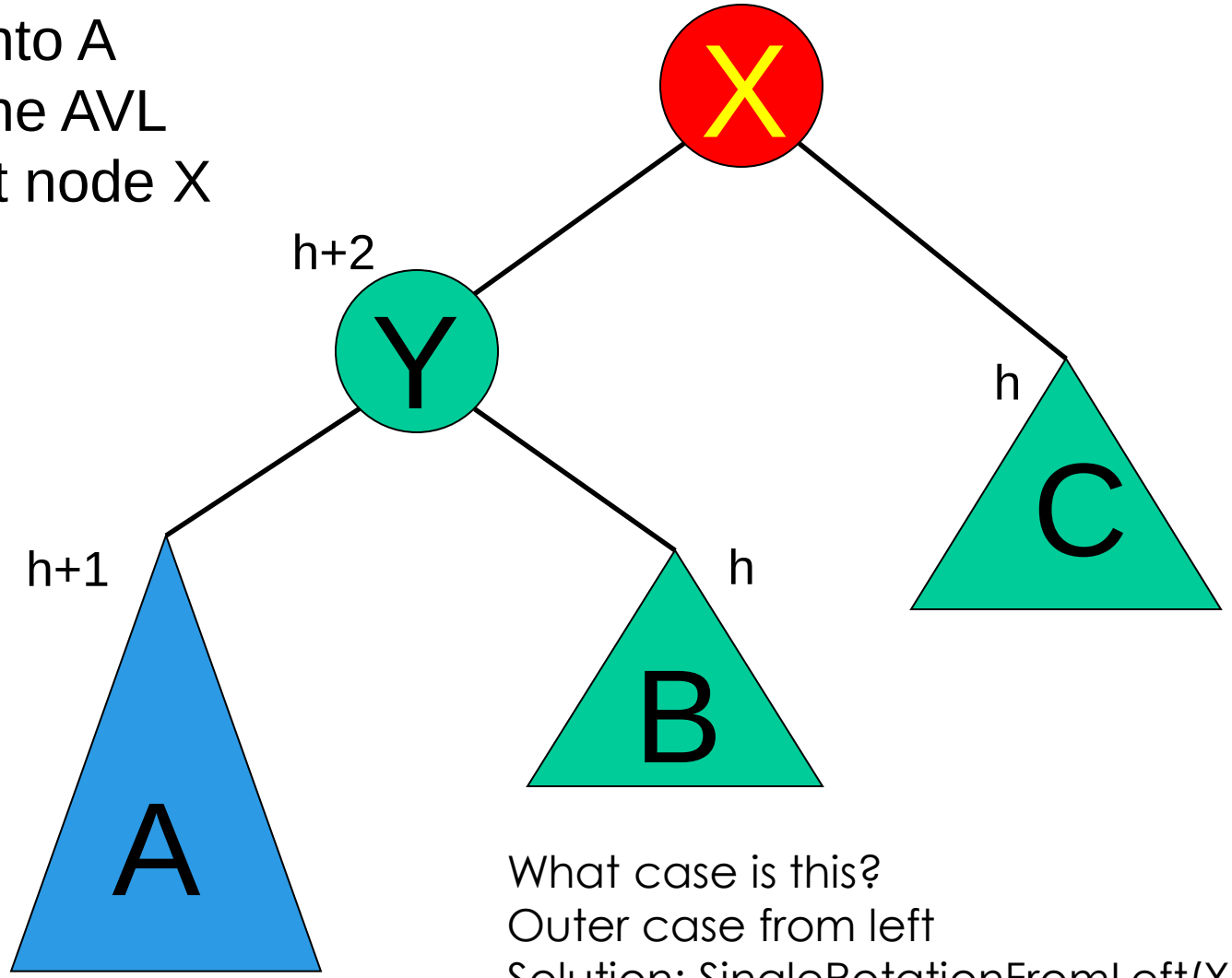
AVL Insertion

Consider a valid AVL subtree



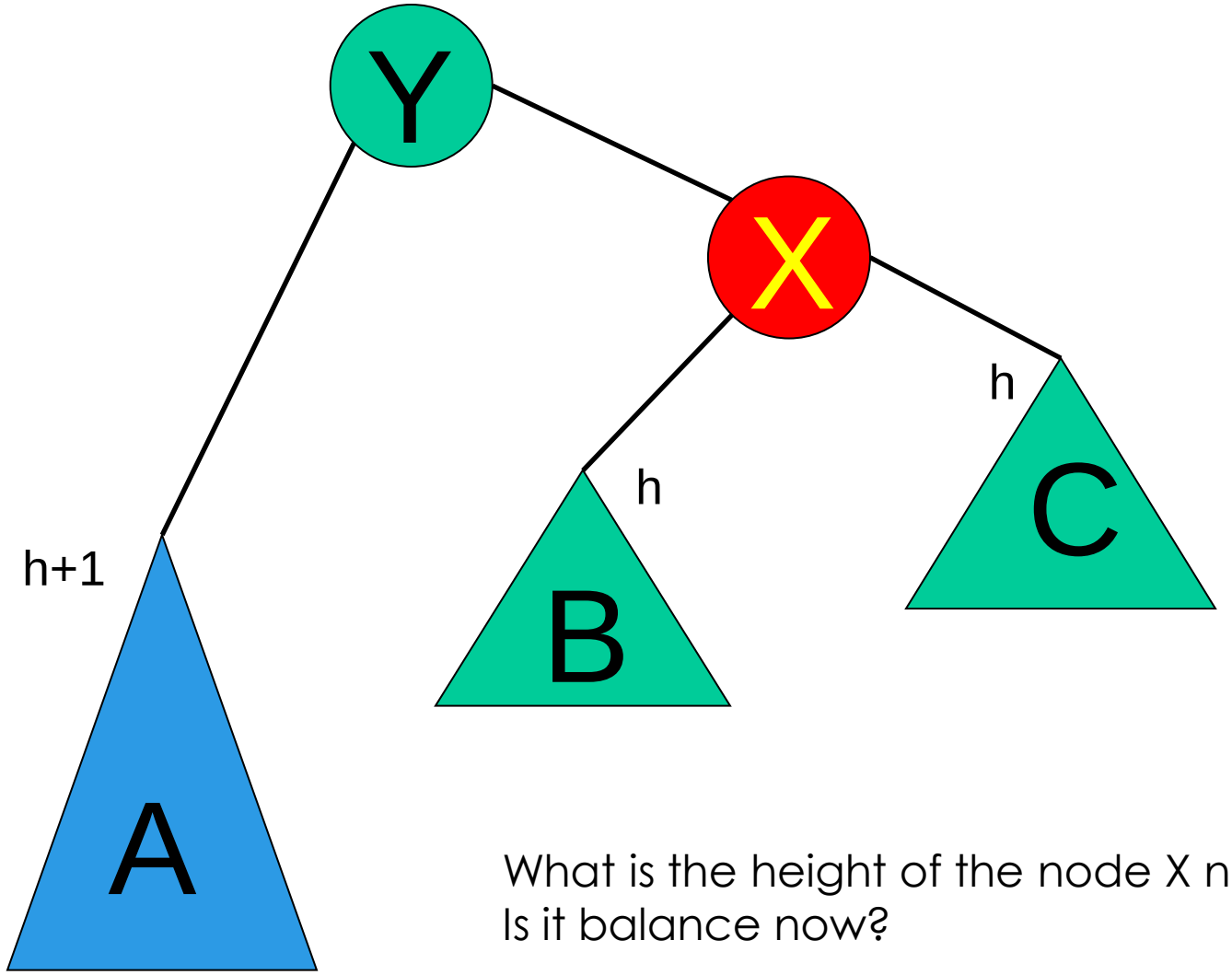
AVL Insertion

Inserting into A destroys the AVL property at node X



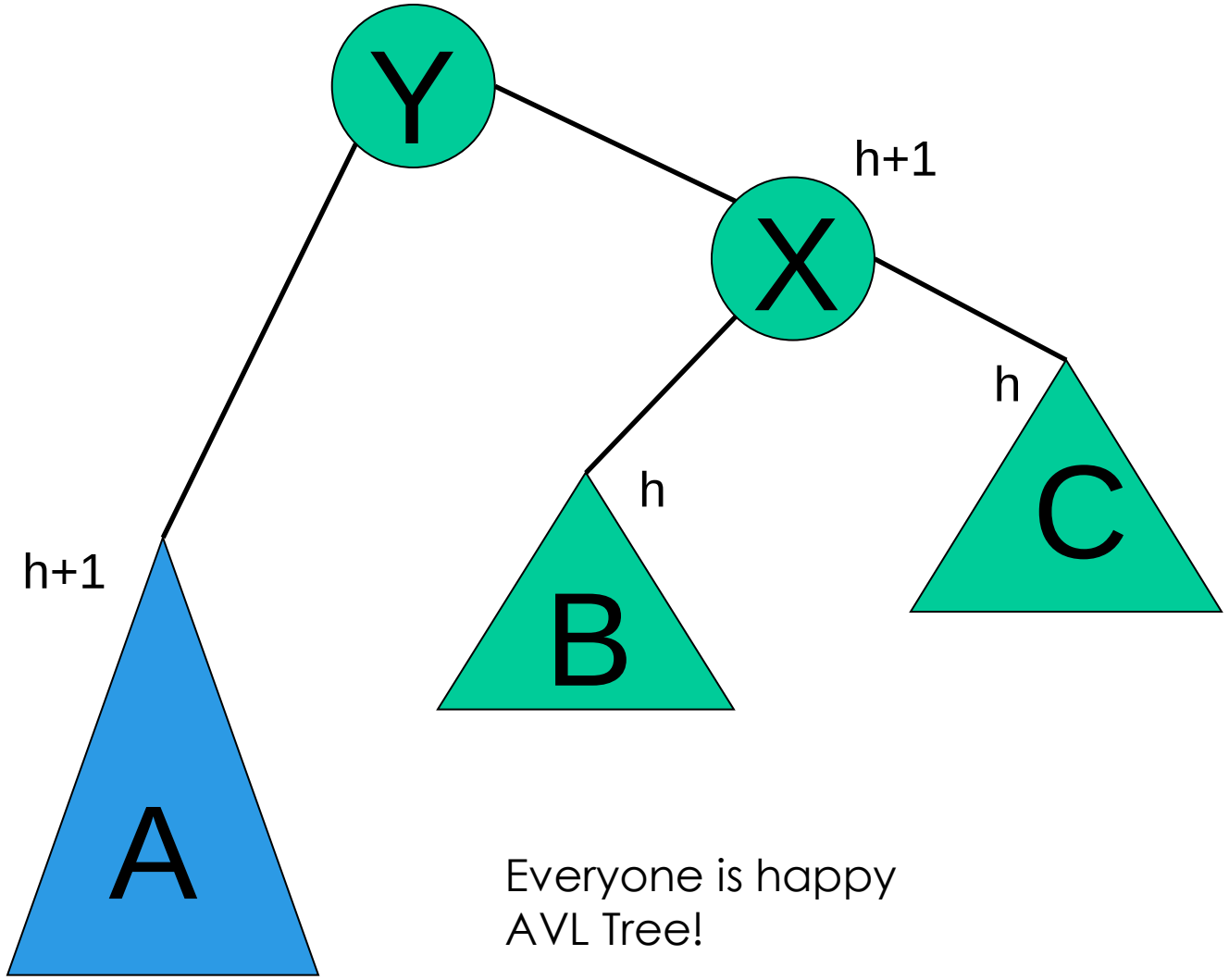
What case is this?
Outer case from left
Solution: SingleRotationFromLeft(X)

SingleRotationFromLeft(X)



What is the height of the node X now?
Is it balance now?

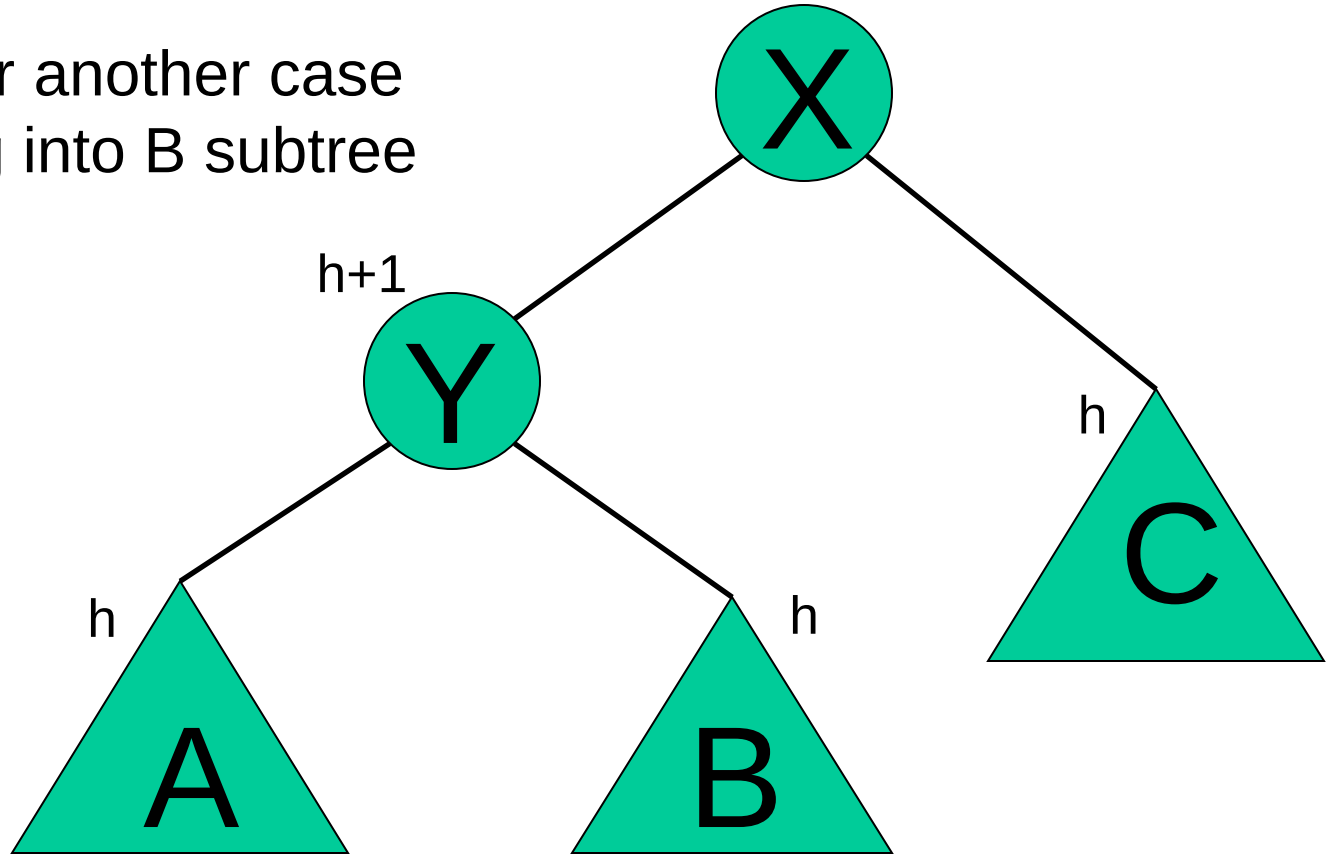
SingleRotationFromLeft(X)



Everyone is happy
AVL Tree!

AVL Insertion

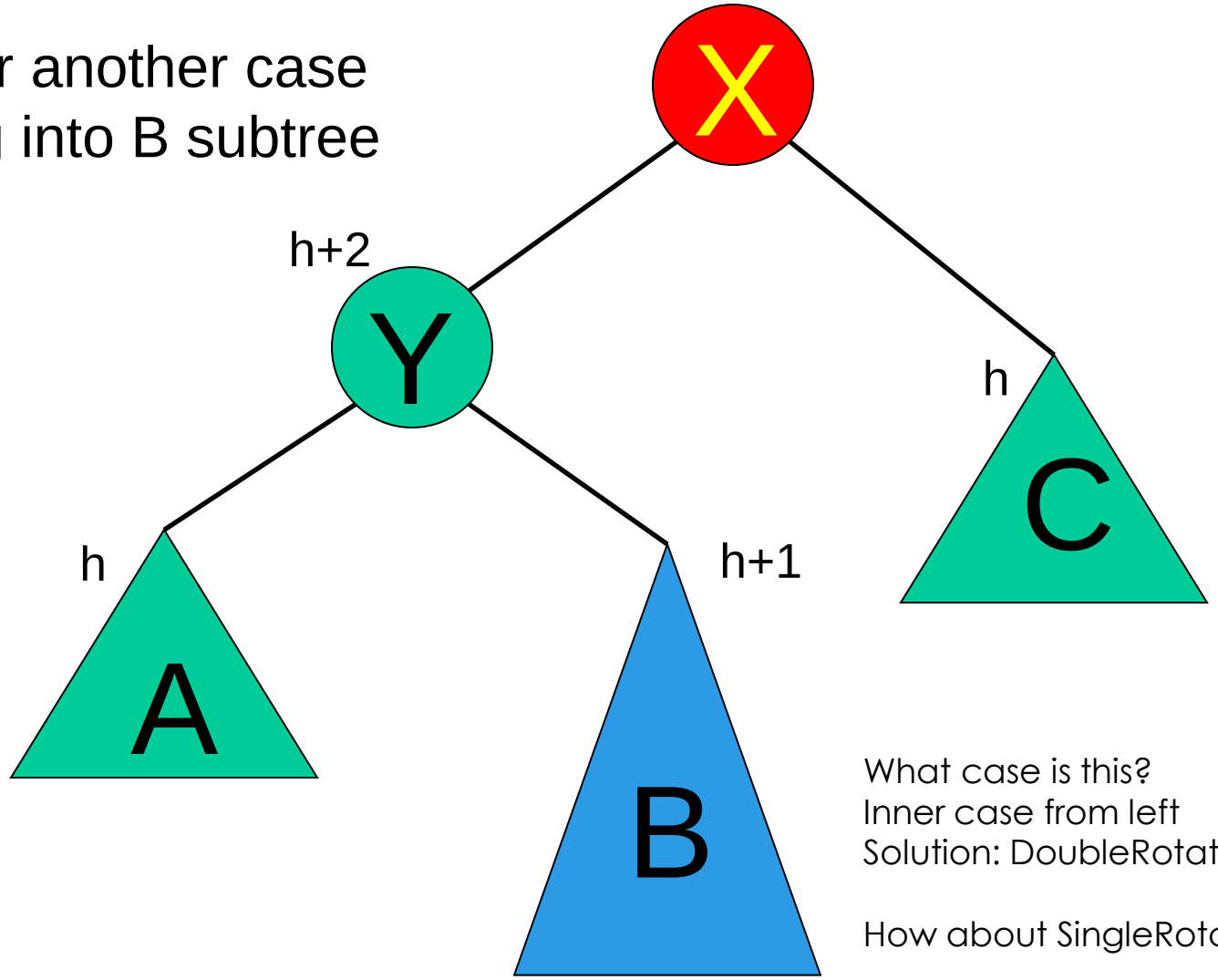
Consider another case
Inserting into B subtree



Inserting a node into the subtree B,
Is there any problem?
What node is imbalance?

AVL Insertion

Consider another case
Inserting into B subtree

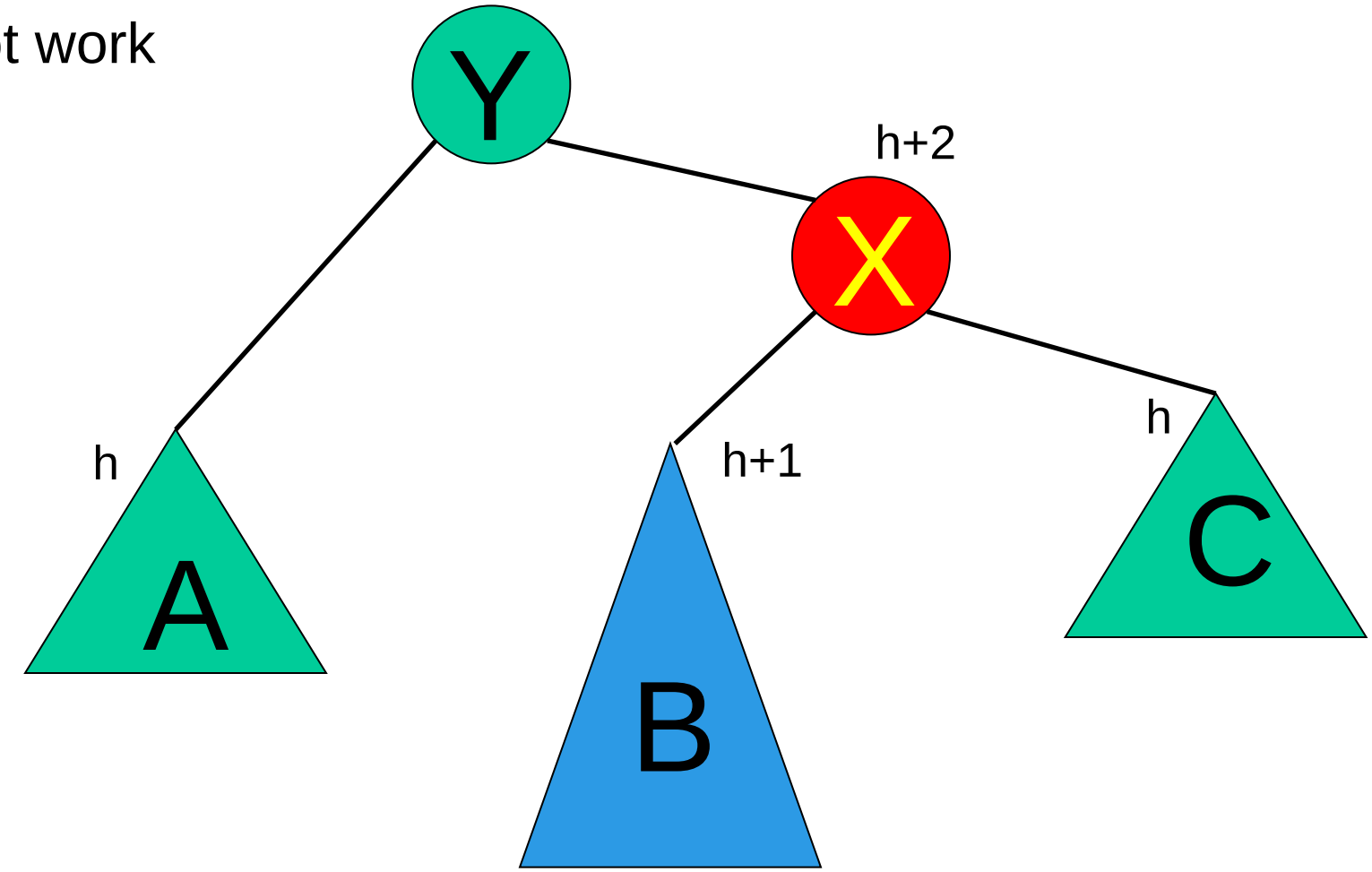


What case is this?
Inner case from left
Solution: DoubleRotationFromLeft(X)

How about SingleRotationFromLeft(X)?

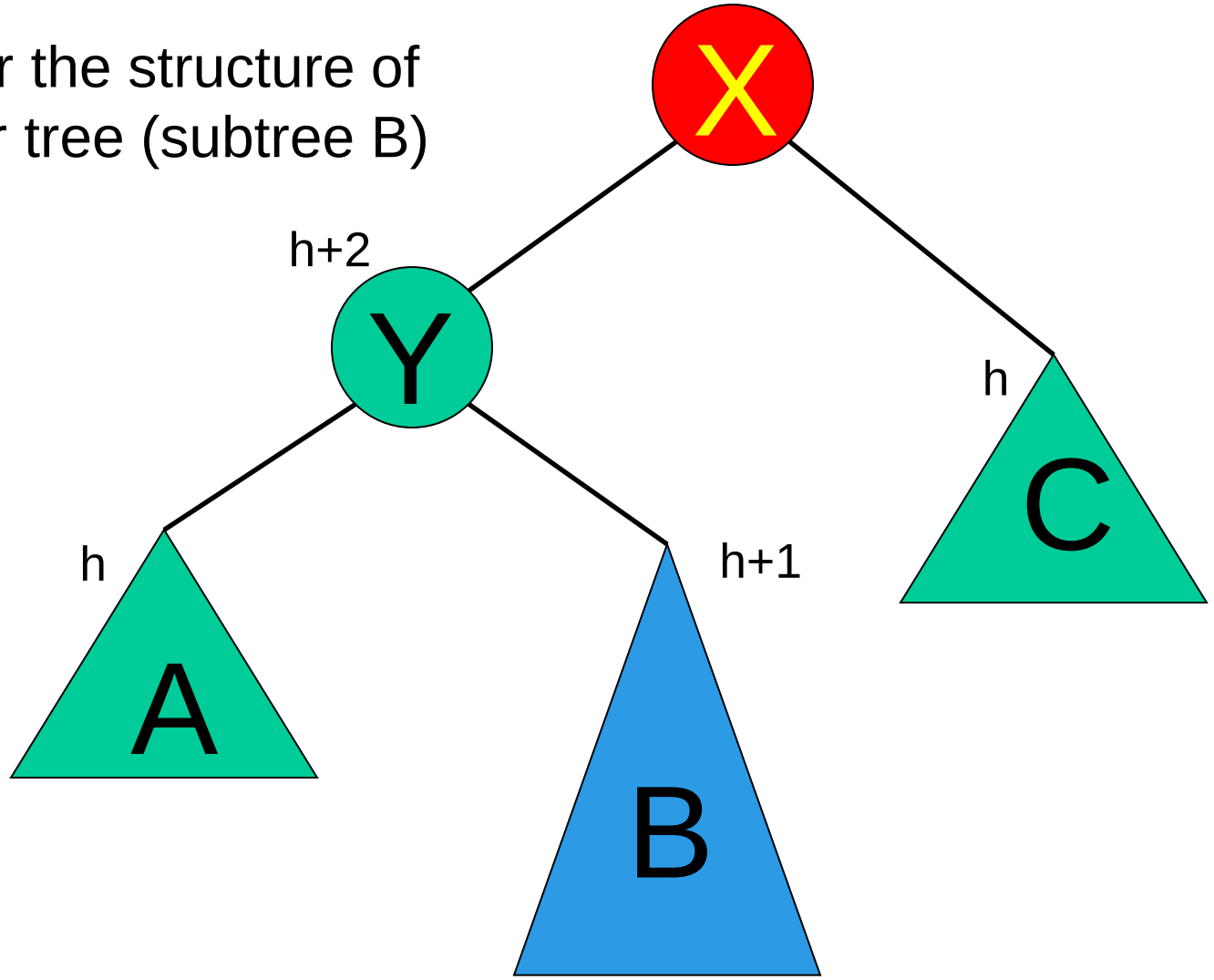
Single Rotation for the Inner Case is Wrong

SingleRotation
does not work



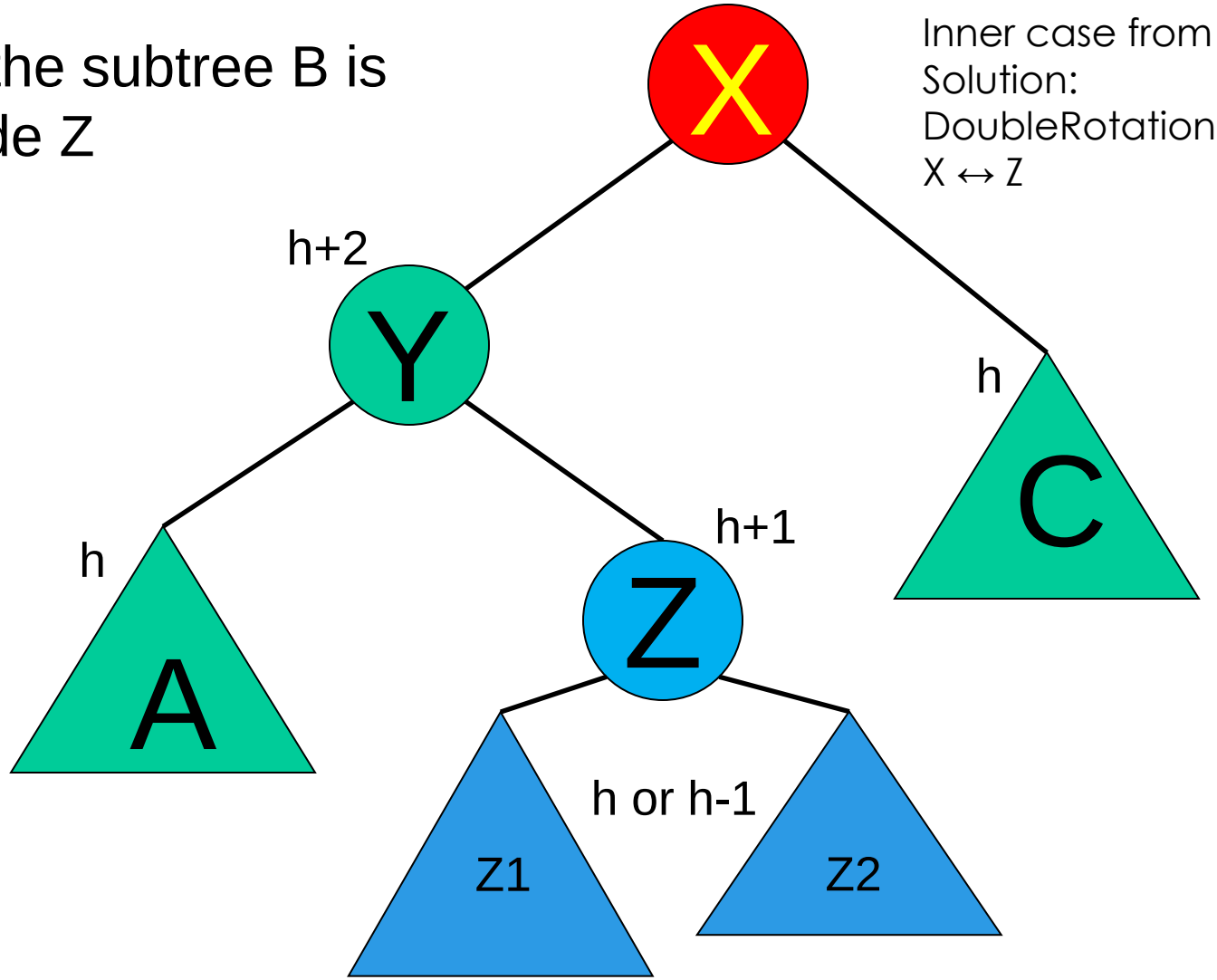
AVL Insertion

Consider the structure of the inner tree (subtree B)



AVL Insertion

Root of the subtree B is now Node Z

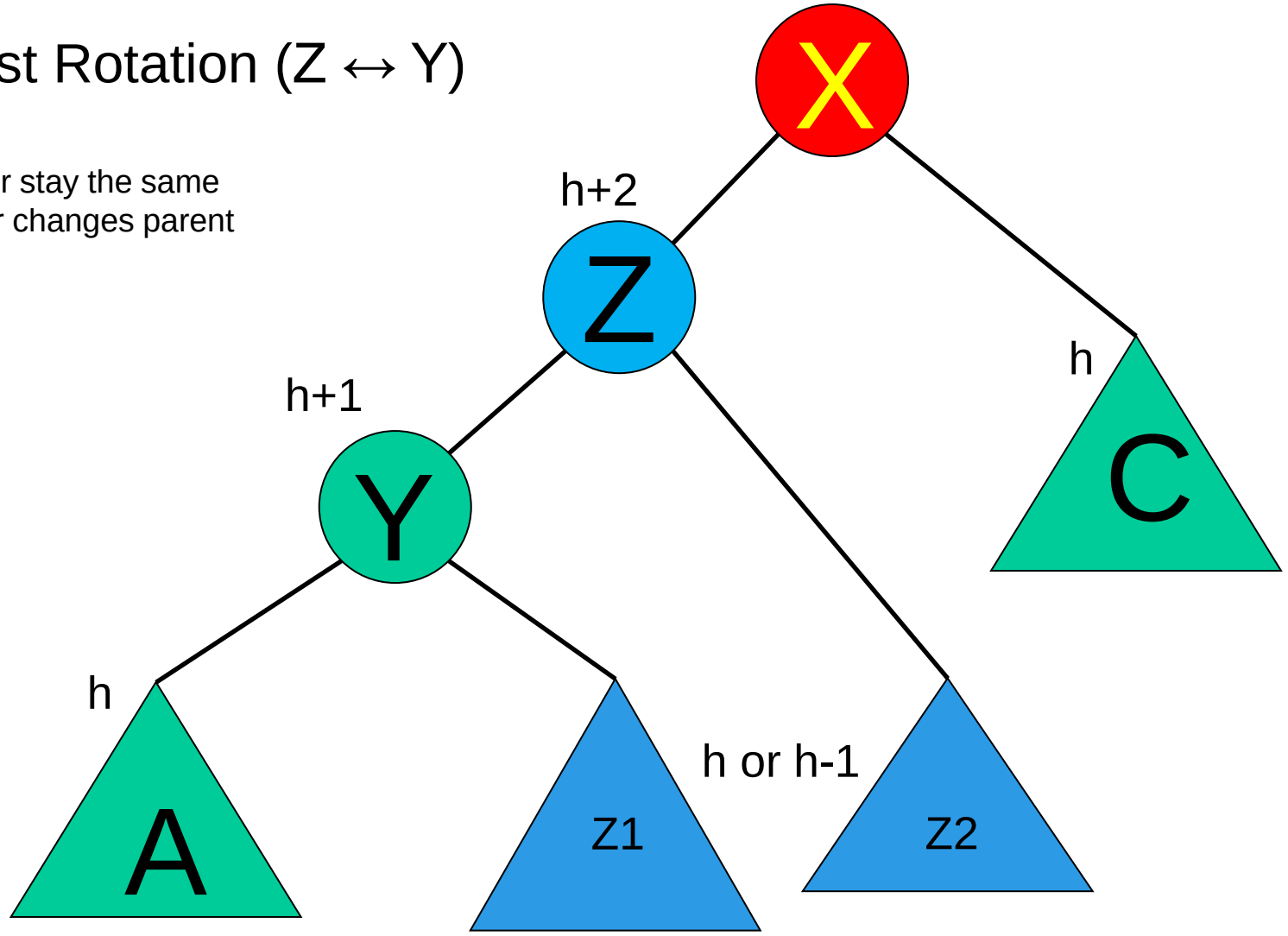


Inner case from left
Solution:
DoubleRotationFromLeft(X)
X ↔ Z

AVL Insertion

First Rotation (Z ↔ Y)

Outer stay the same
Inner changes parent

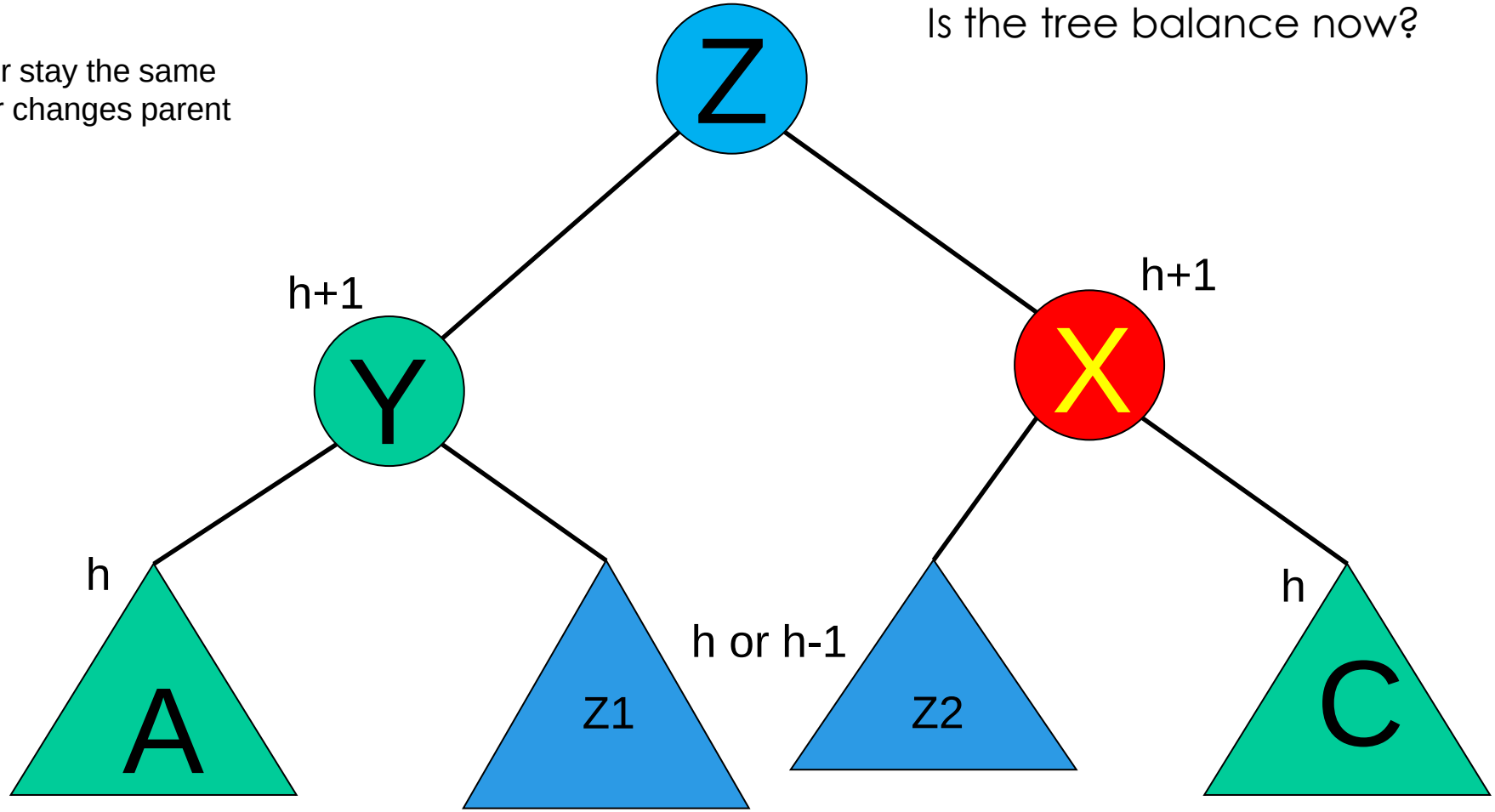


AVL Insertion

Second Rotation (Z ↔ X)

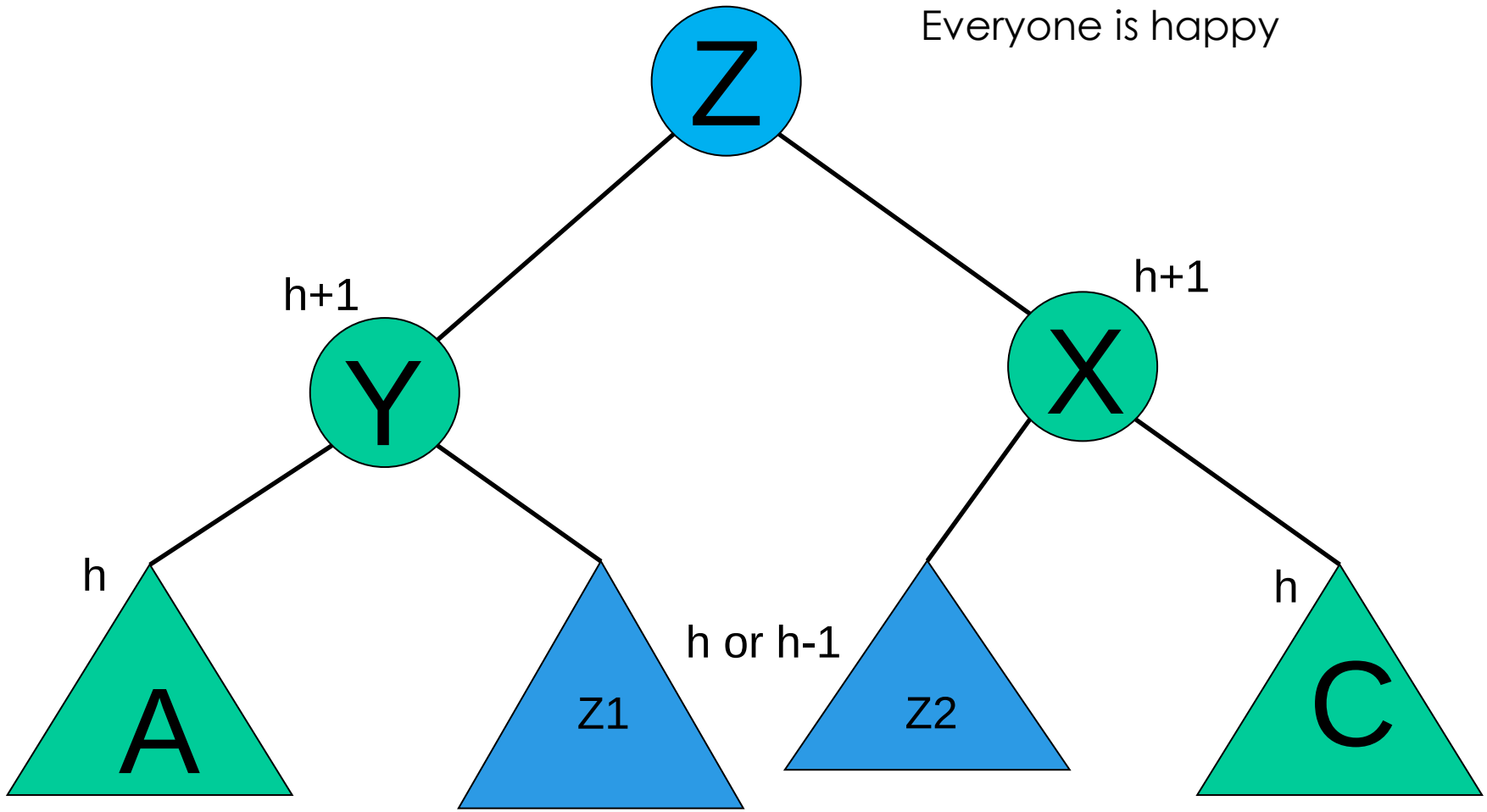
Outer stay the same
Inner changes parent

After DoubleRotation,
Is the tree balance now?

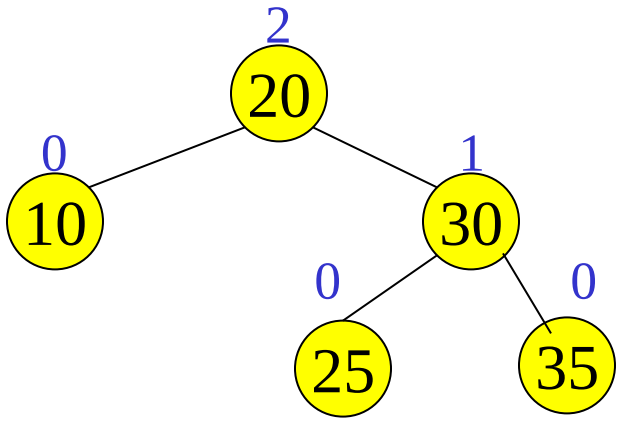


AVL Insertion

Double Rotation



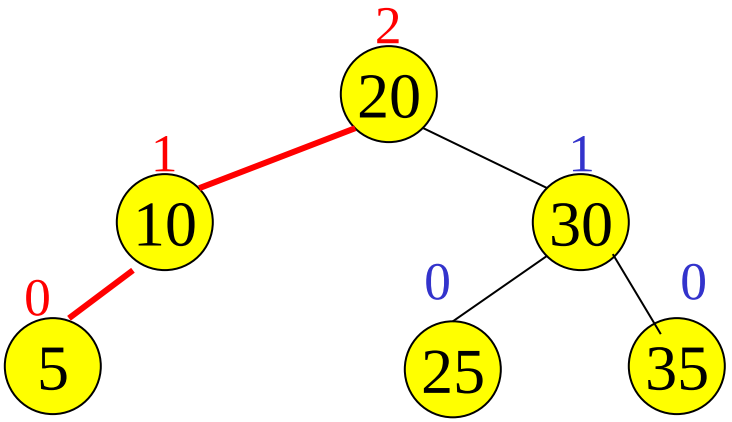
Example

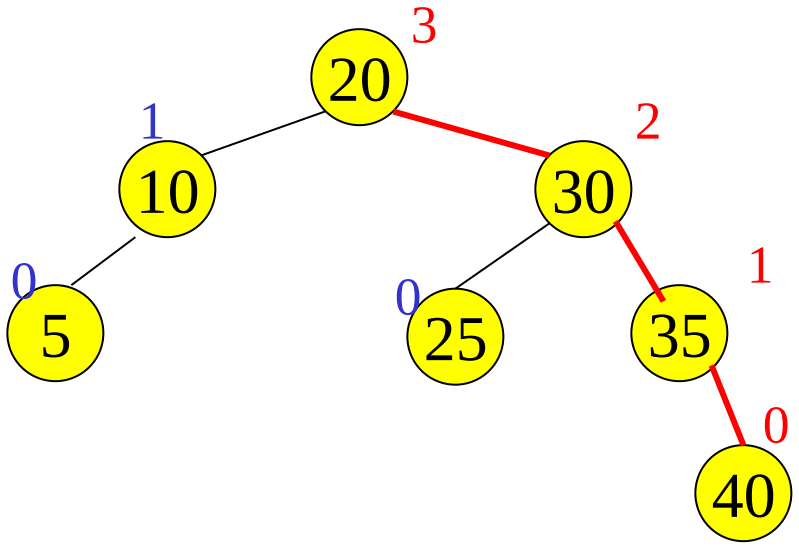


Insert(5)

Insert(40)

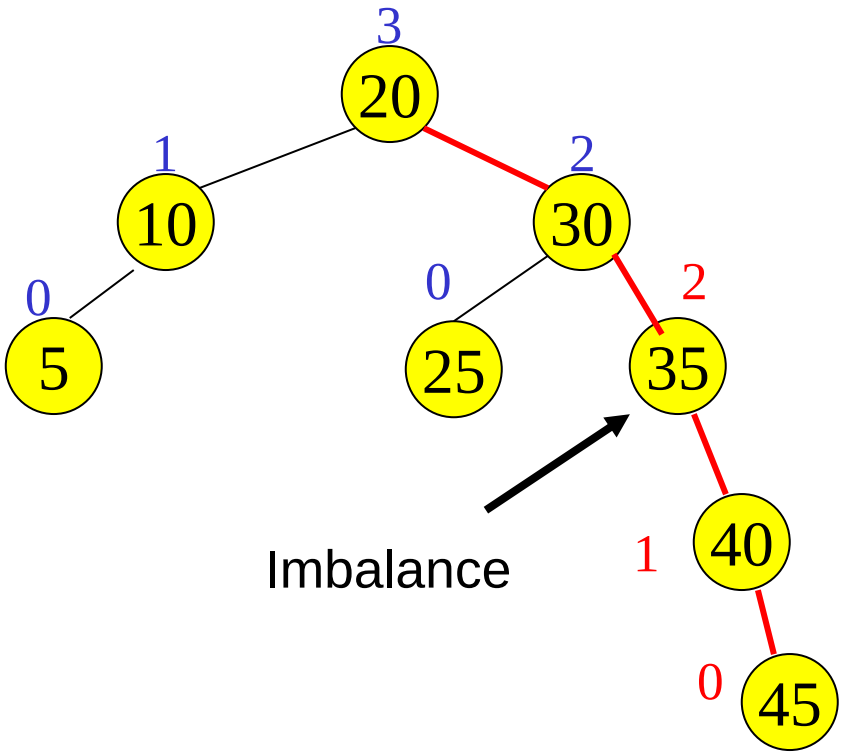
Example

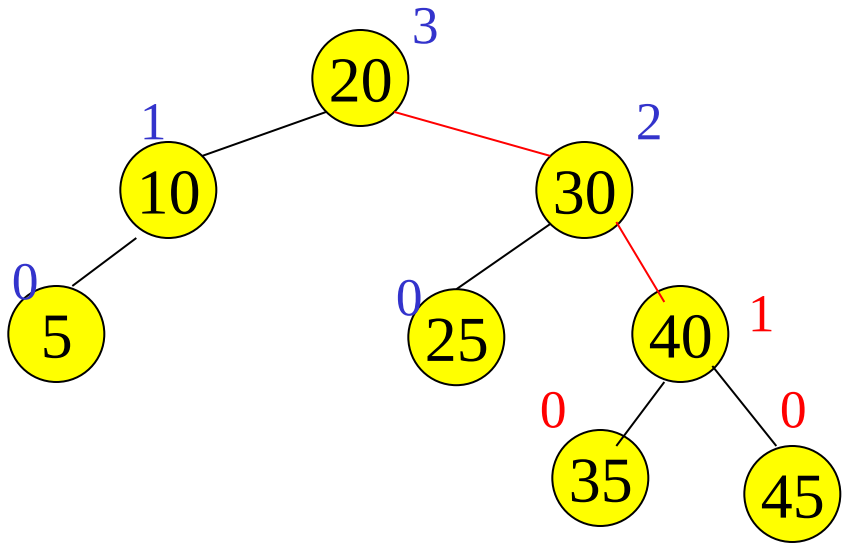




Now Insert(45)

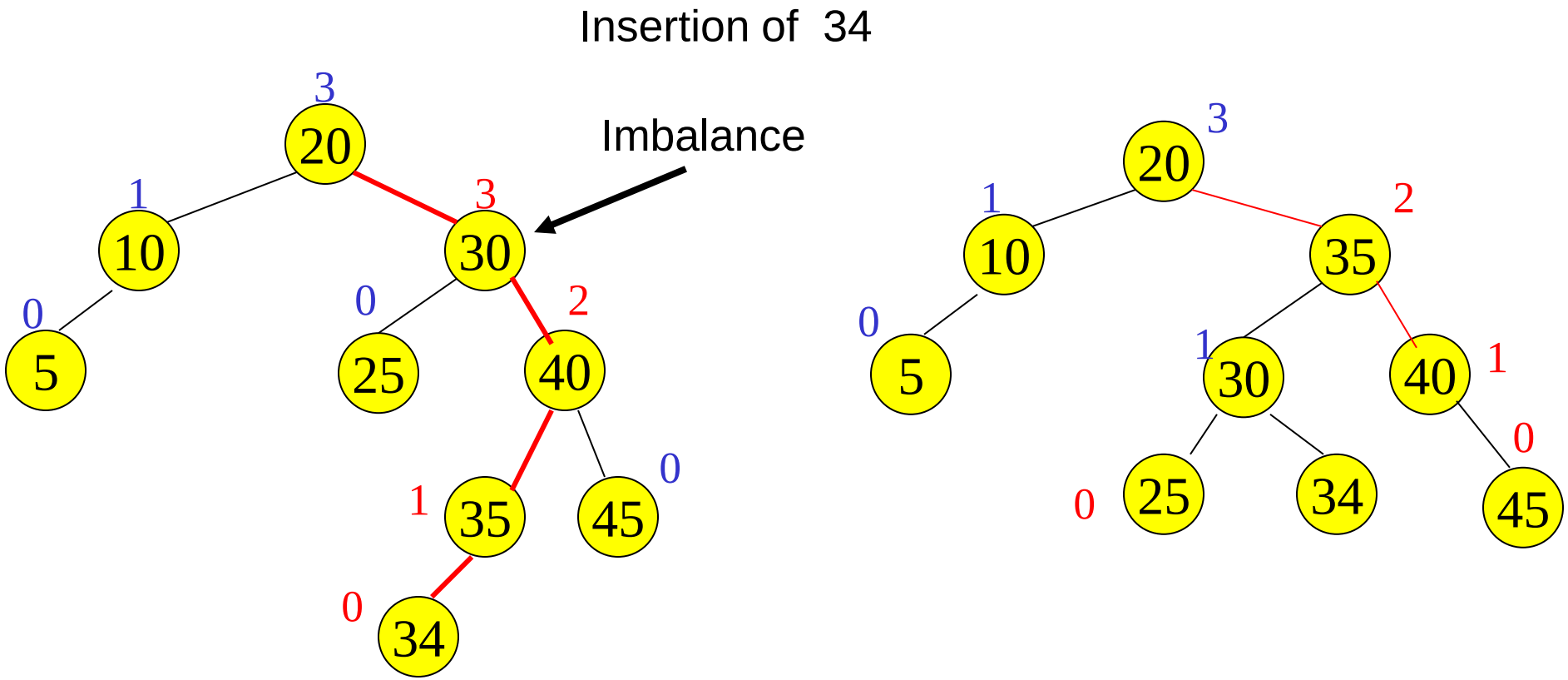
Example





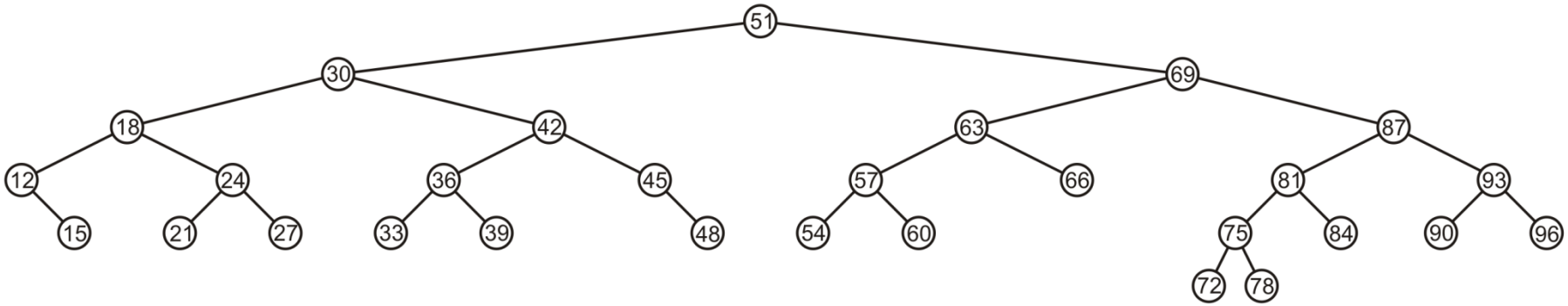
Now Insert(34)

Example


Insertion of 34
Imbalance
20
10
30
5
25
40
35
45
34
3
1
3
0
0
2
1
0
0
20
10
35
5
30
40
25
34
45
3
1
2
0
1
1
0
0

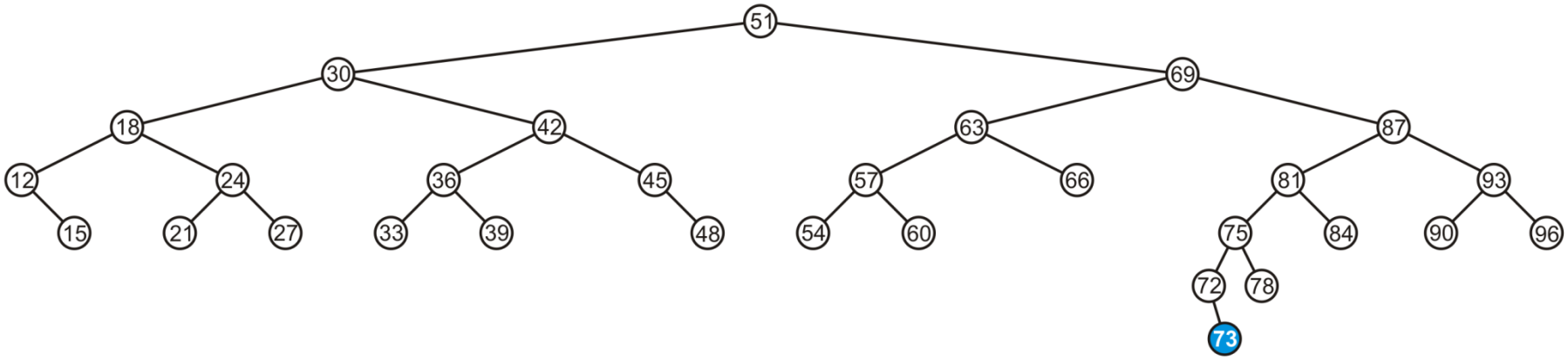
Insertion

Consider this AVL tree



Insertion

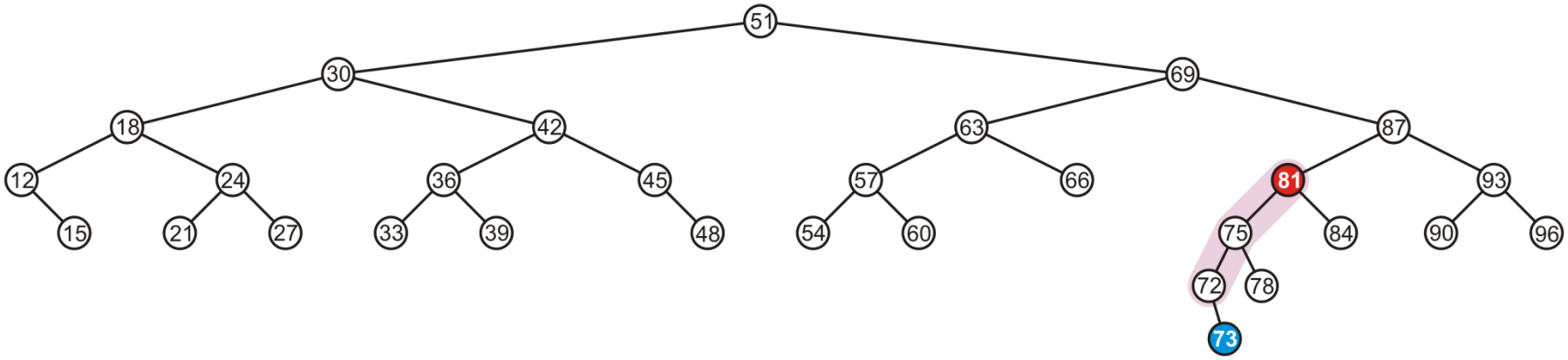
Insert 73



Insertion

The node 81 is unbalanced

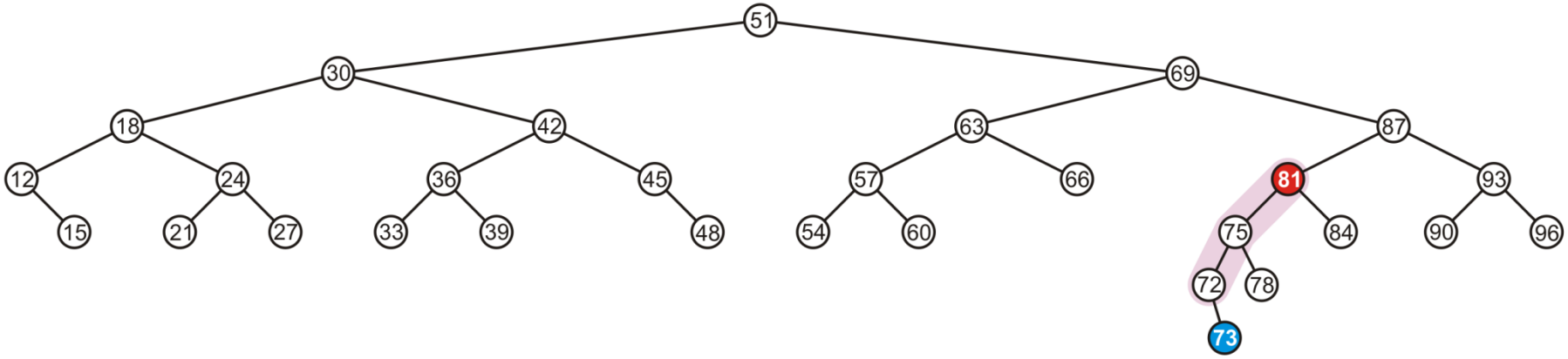
– A left-left imbalance

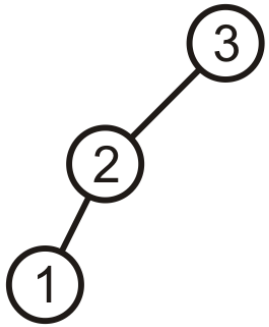


Insertion

The node 81 is unbalanced

– A left-left imbalance

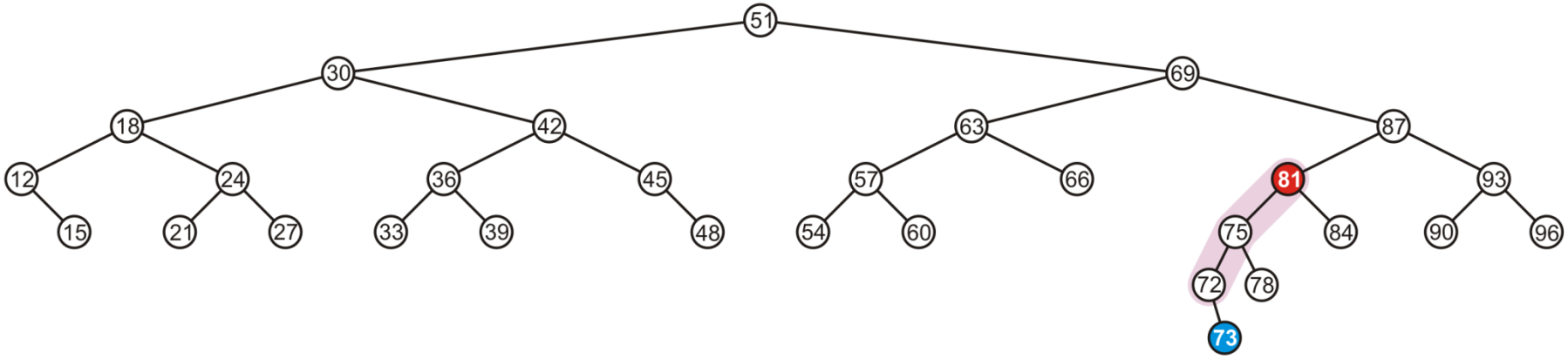


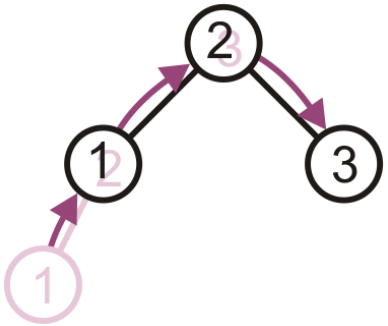


Insertion

The node 81 is unbalanced

- A left-left imbalance
- Promote the intermediate node to the imbalanced node

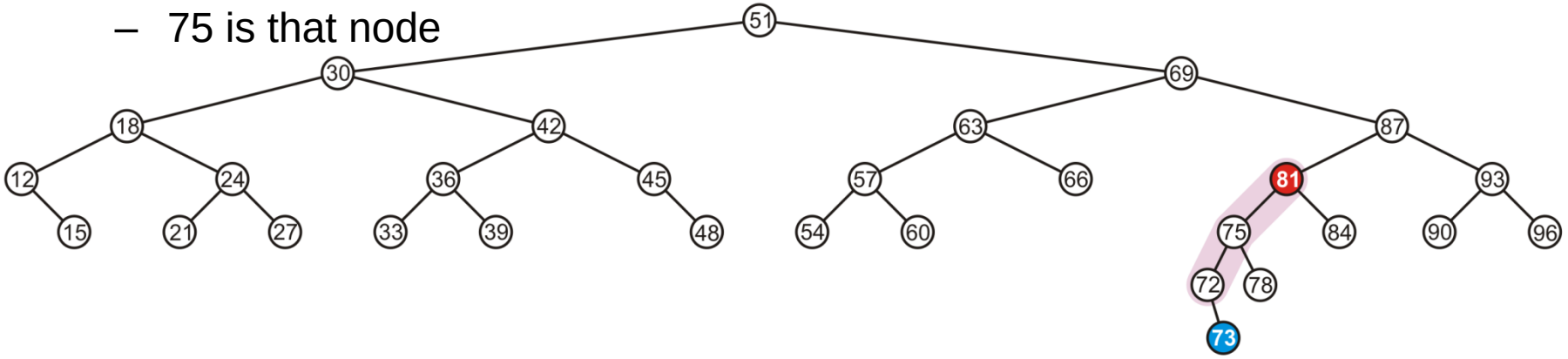


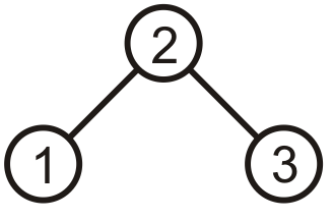


Insertion

The node 81 is unbalanced

- A left-left imbalance
- Promote the intermediate node to the imbalanced node
- 75 is that node

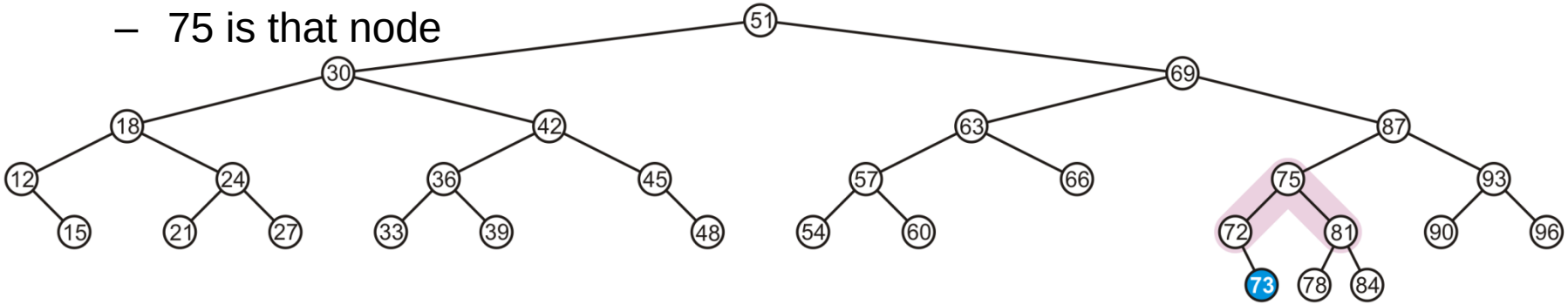




Insertion

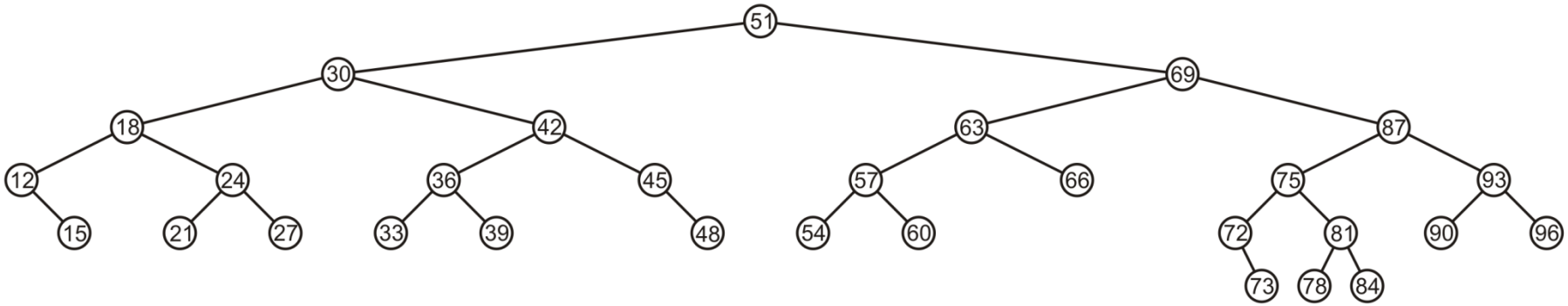
The node 81 is unbalanced

- A left-left imbalance
- Promote the intermediate node to the imbalanced node
- 75 is that node



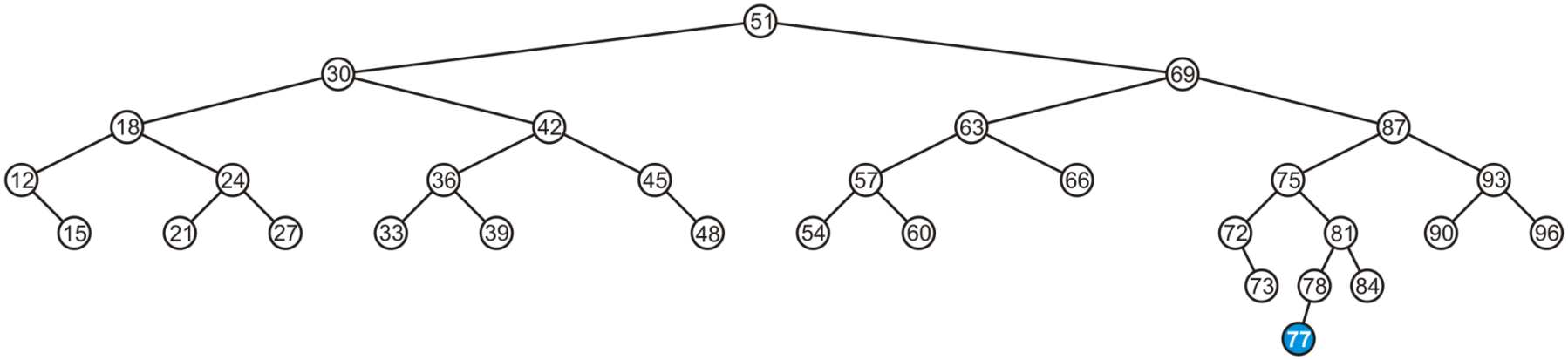
Insertion

The tree is AVL balanced



Insertion

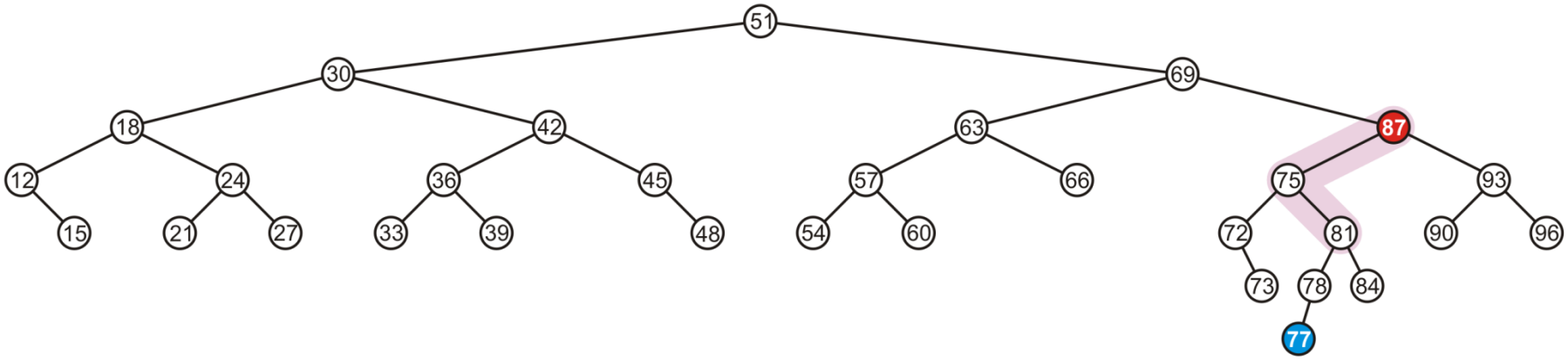
Insert 77



Insertion

The node 87 is unbalanced

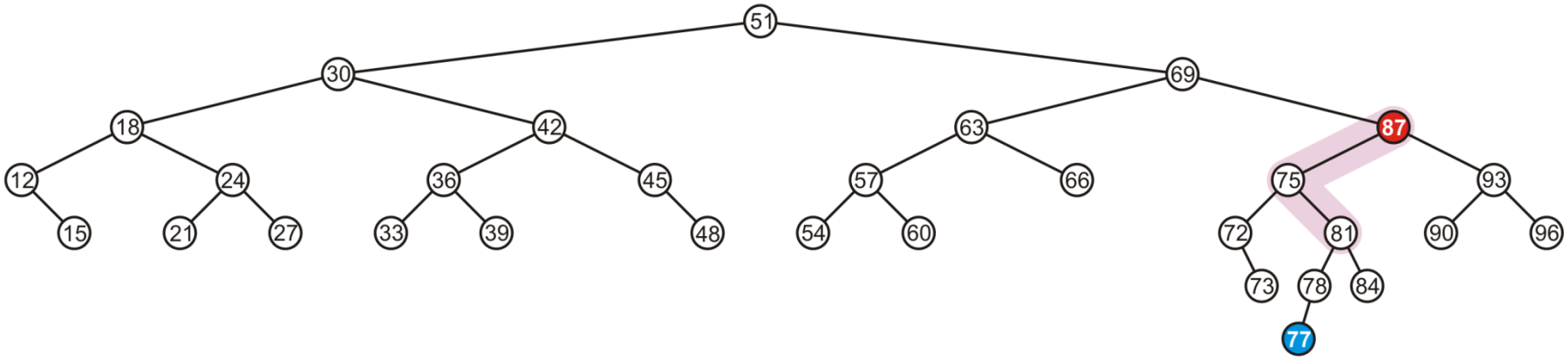
– A left-right imbalance

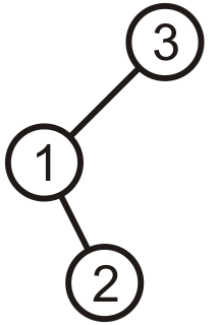


Insertion

The node 87 is unbalanced

– A left-right imbalance

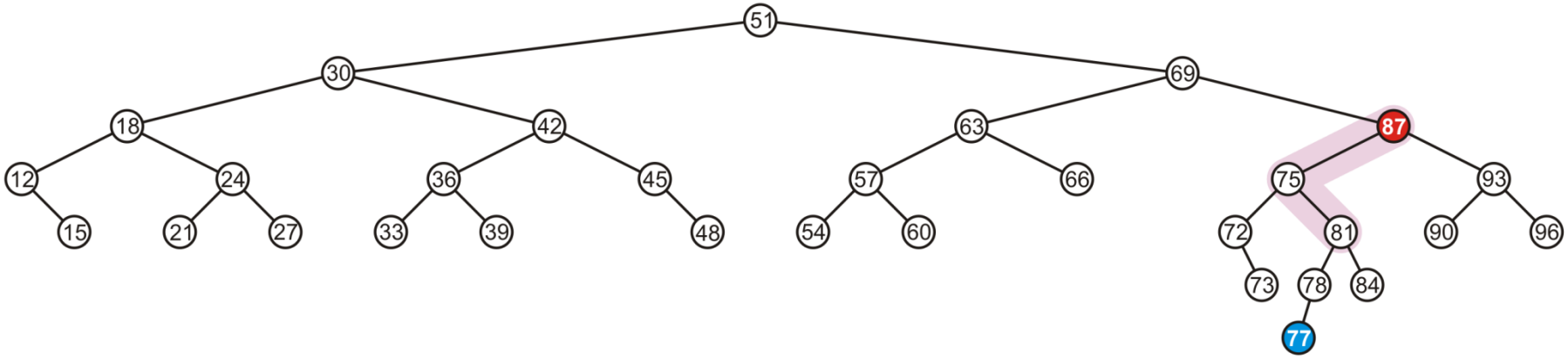


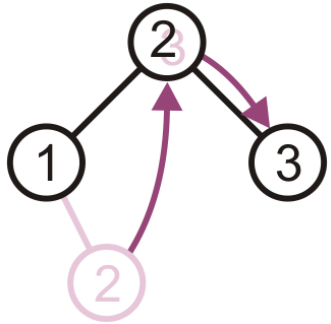


Insertion

The node 87 is unbalanced

- A left-right imbalance
- Promote the intermediate node to the imbalanced node

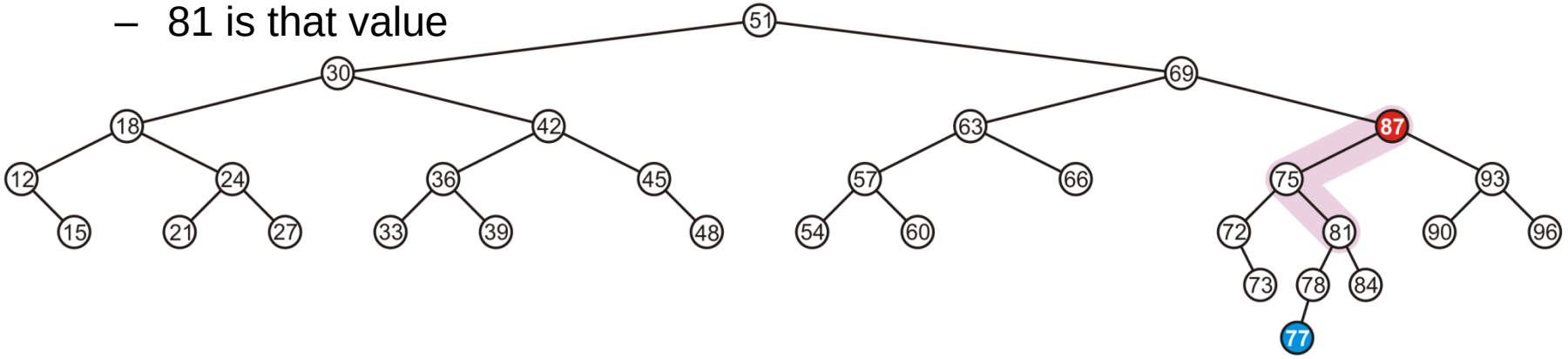


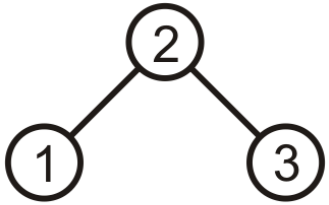


Insertion

The node 87 is unbalanced

- A left-right imbalance
- Promote the intermediate node to the imbalanced node
- 81 is that value

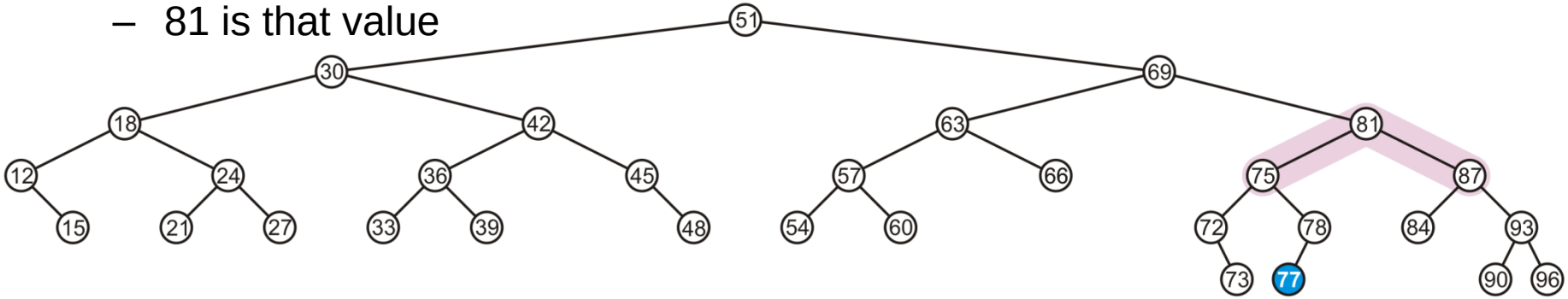




Insertion

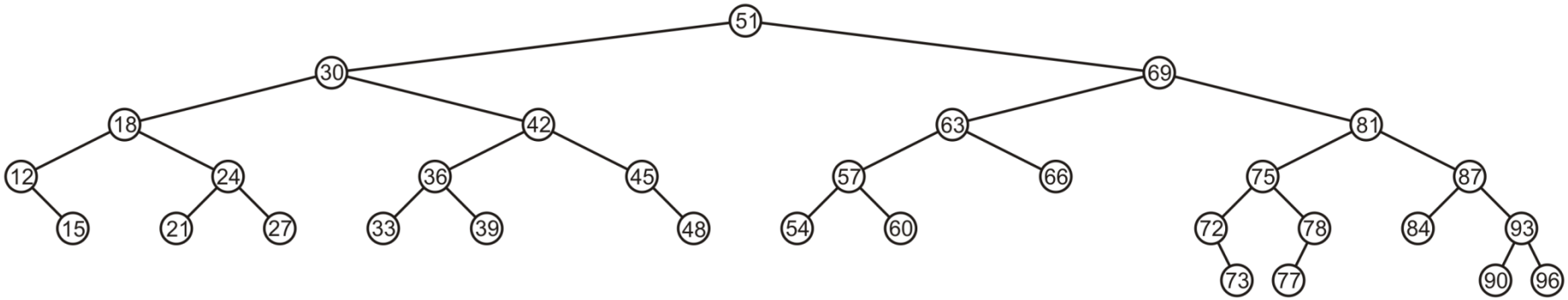
The node 87 is unbalanced

- A left-right imbalance
- Promote the intermediate node to the imbalanced node
- 81 is that value



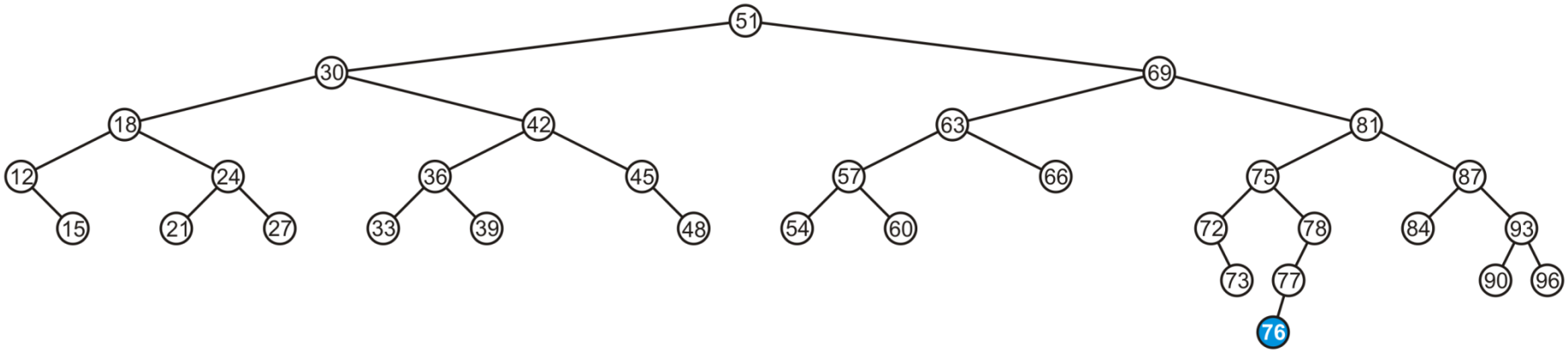
Insertion

The tree is balanced



Insertion

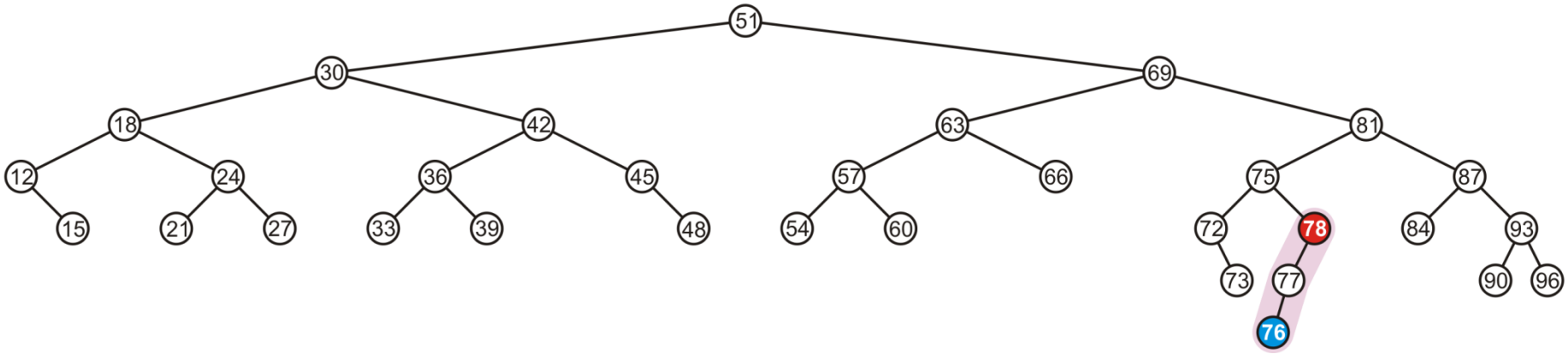
Insert 76



Insertion

The node 78 is unbalanced

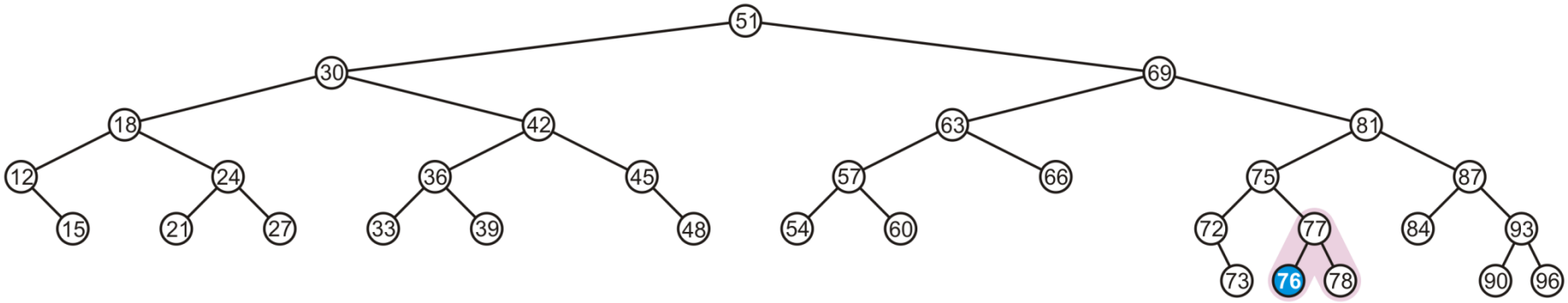
– A left-left imbalance



Insertion

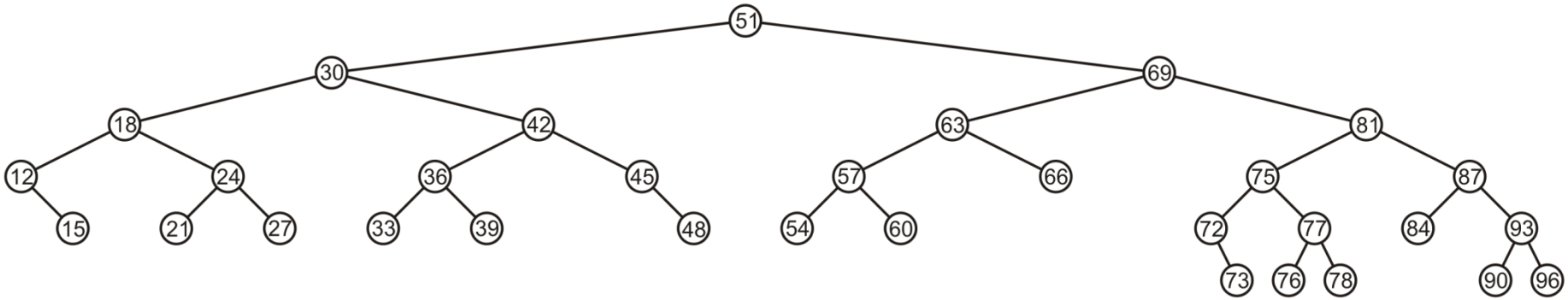
The node 78 is unbalanced

– Promote 77



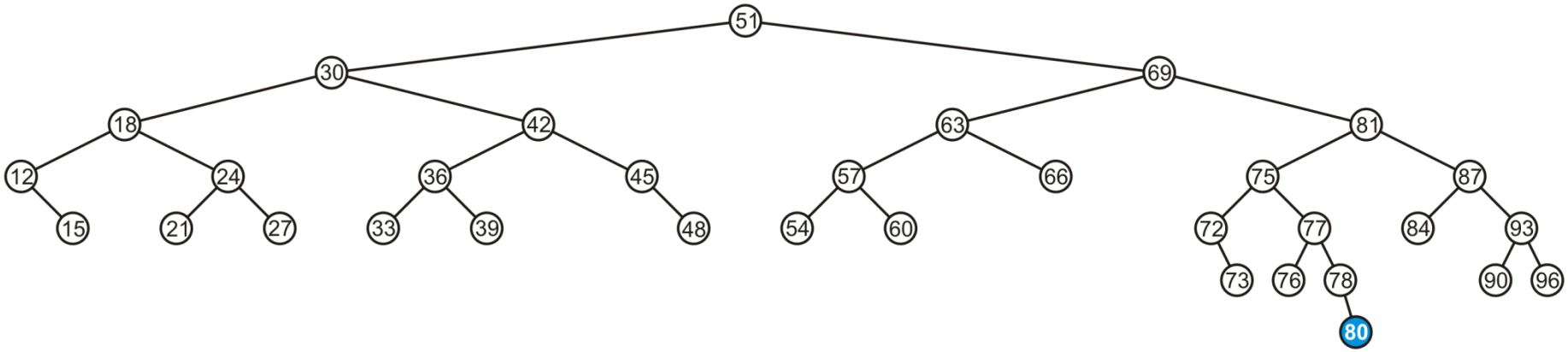
Insertion

Again, balanced



Insertion

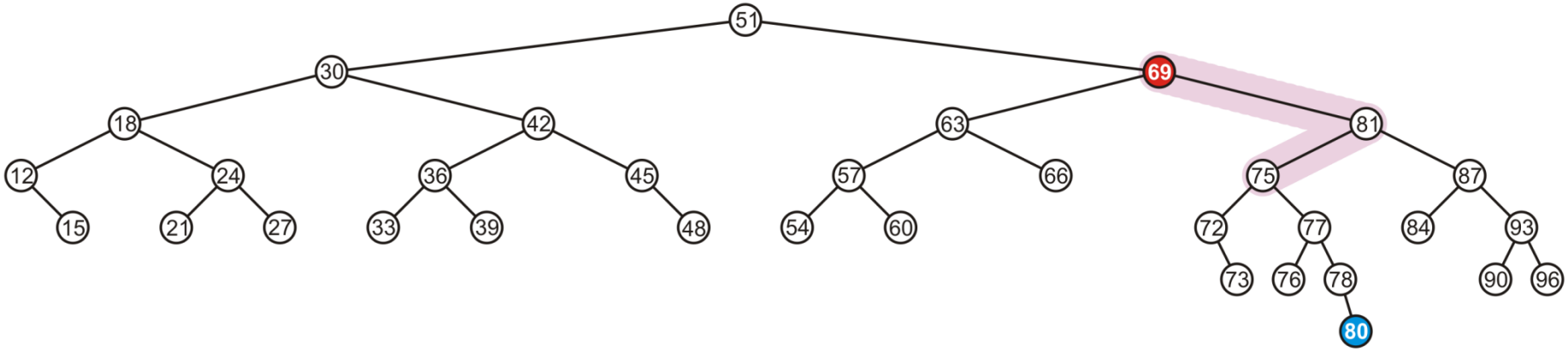
Insert 80



Insertion

The node 69 is unbalanced

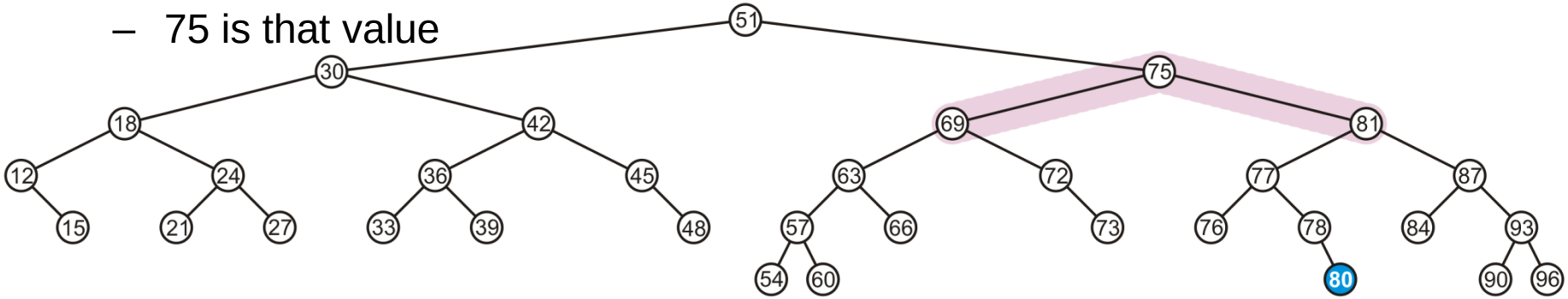
- A right-left imbalance
- Promote the intermediate node to the imbalanced node



Insertion

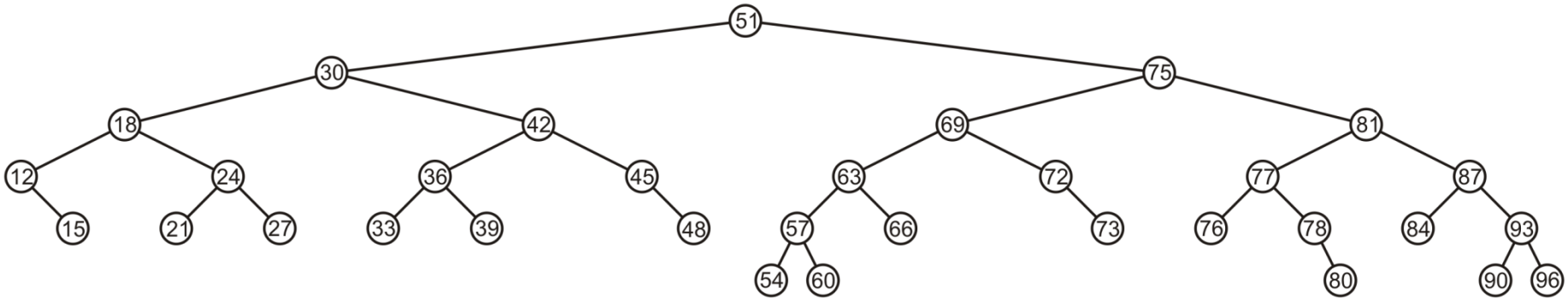
The node 69 is unbalanced

- A left-right imbalance
- Promote the intermediate node to the imbalanced node
- 75 is that value



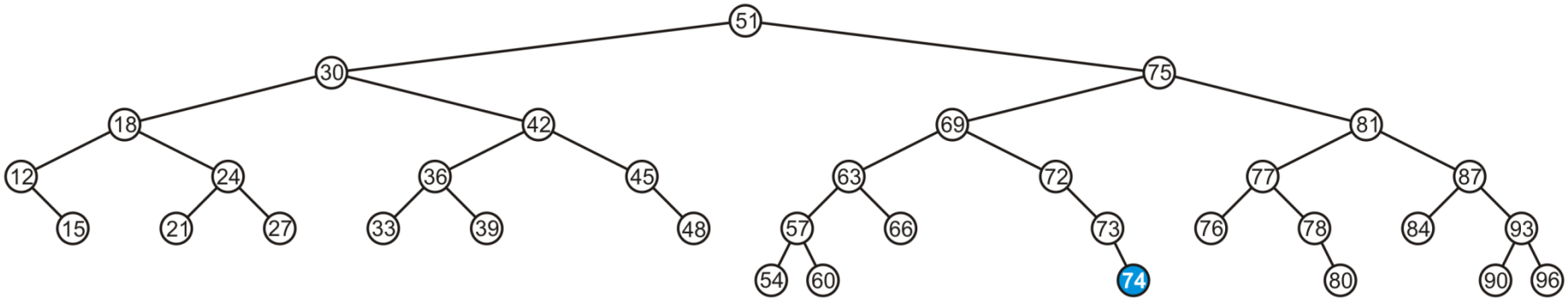
Insertion

Again, balanced



Insertion

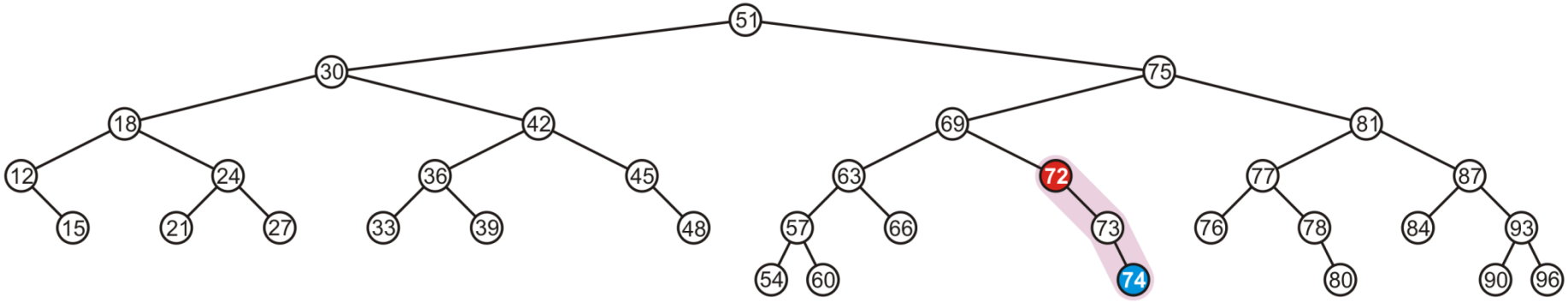
Insert 74



Insertion

The node 72 is unbalanced

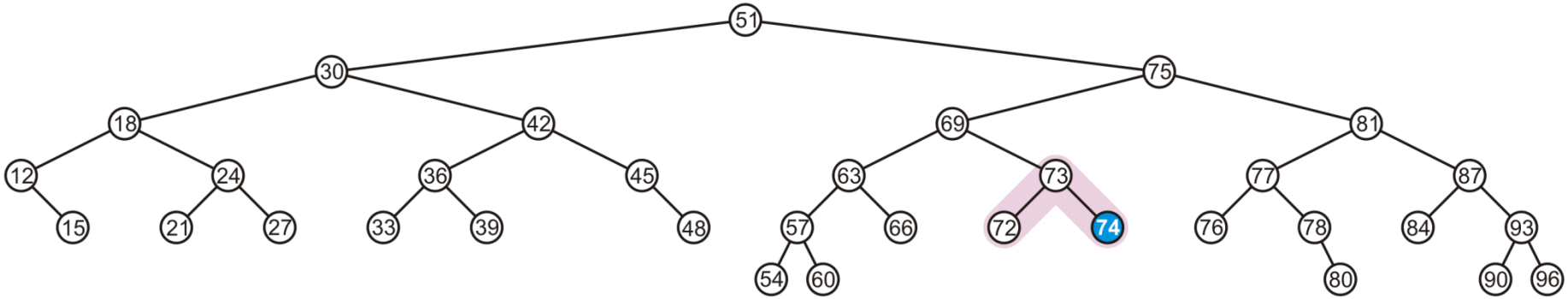
- A right-right imbalance
- Promote the intermediate node to the imbalanced node



Insertion

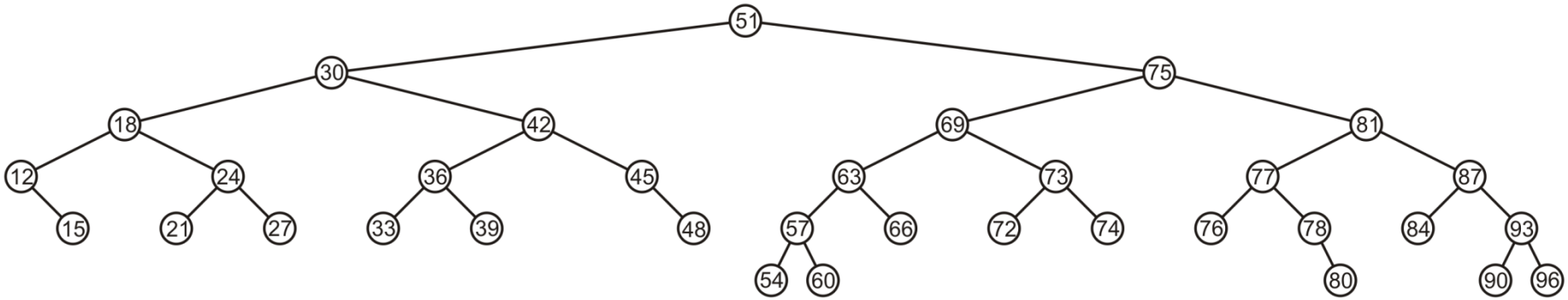
The node 72 is unbalanced

- A right-right imbalance
- Promote the intermediate node to the imbalanced node



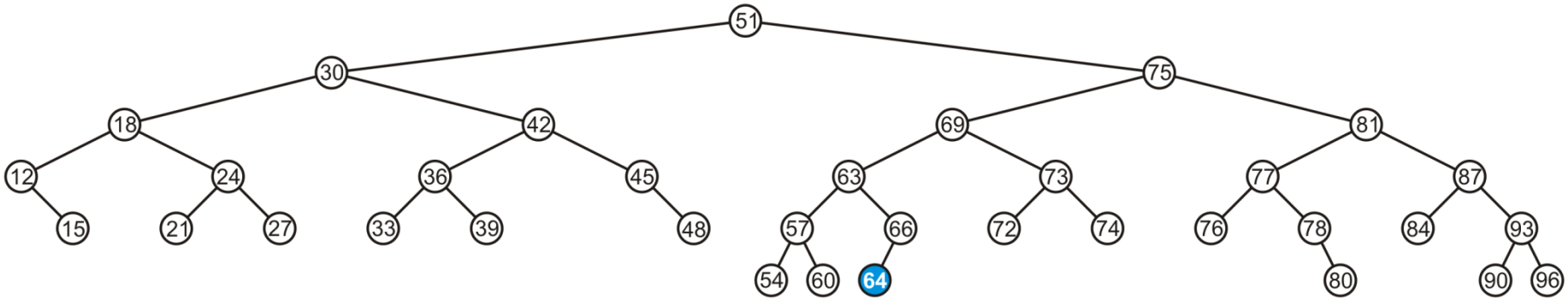
Insertion

Again, balanced



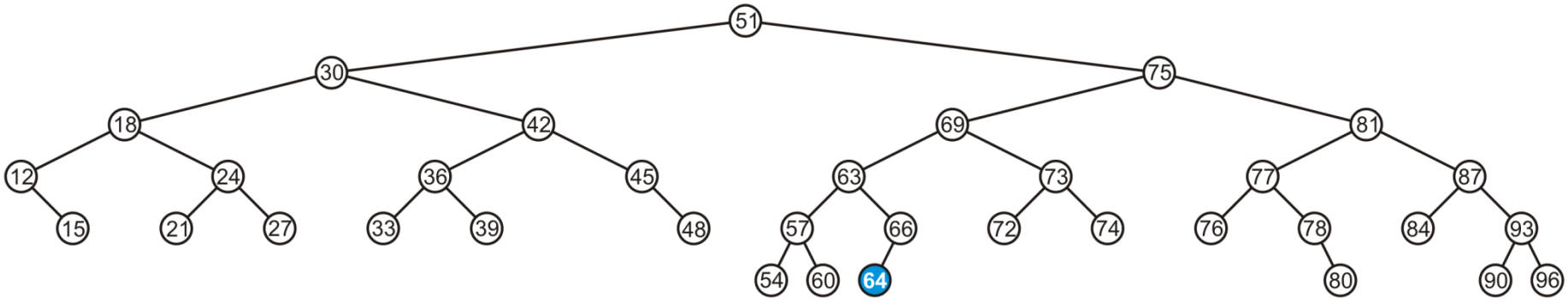
Insertion

Insert 64



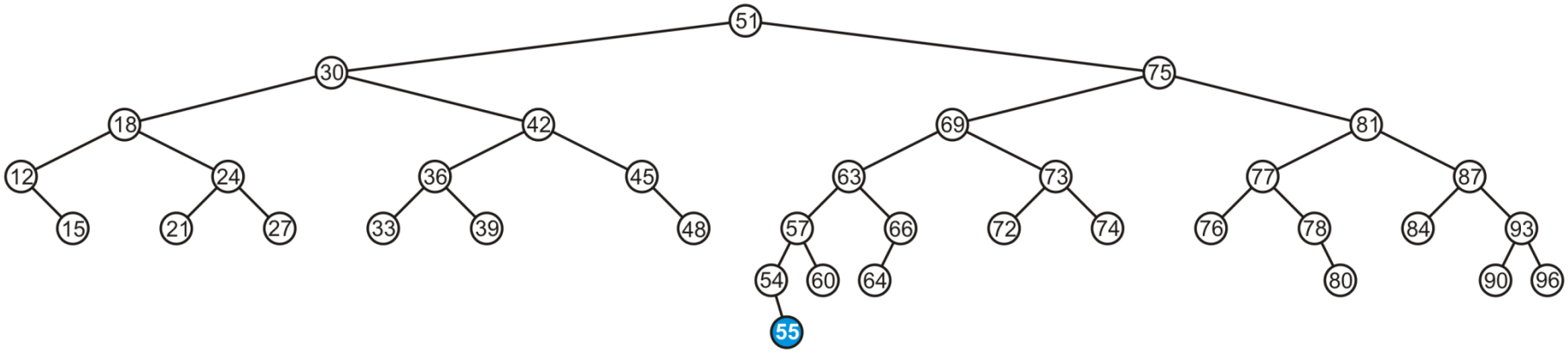
Insertion

This causes no imbalances



Insertion

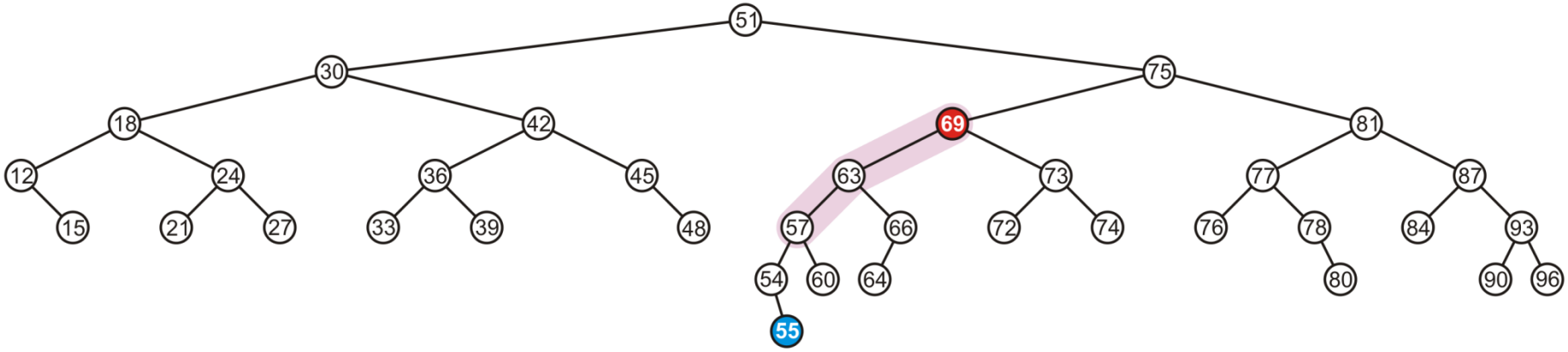
Insert 55



Insertion

The node 69 is imbalanced

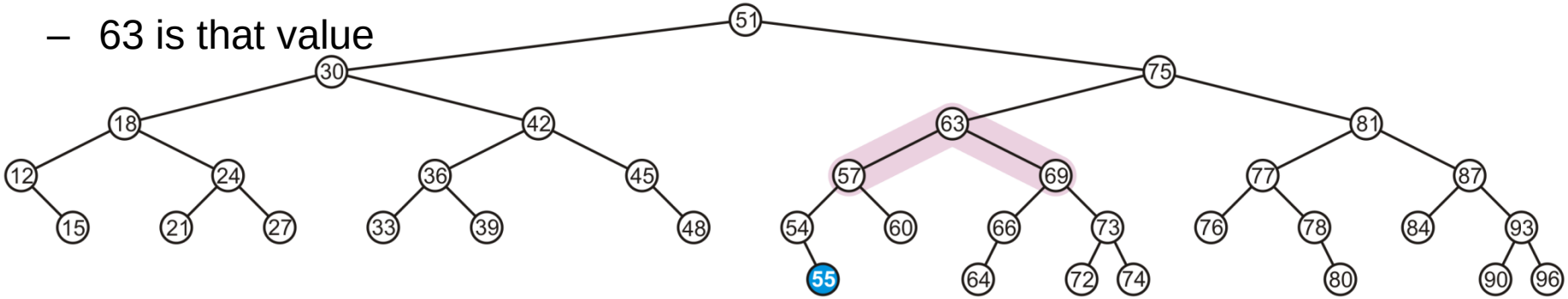
- A left-left imbalance
- Promote the intermediate node to the imbalanced node



Insertion

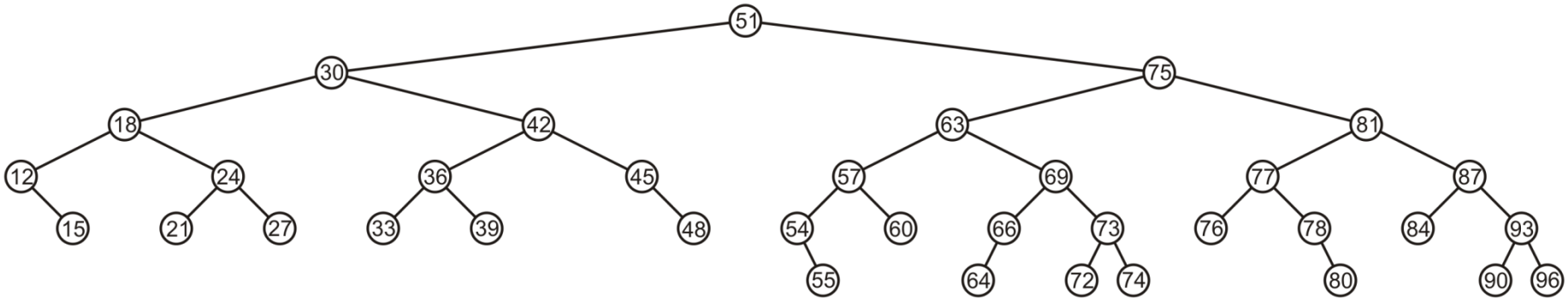
The node 69 is imbalanced

- A left-left imbalance
- Promote the intermediate node to the imbalanced node
- 63 is that value



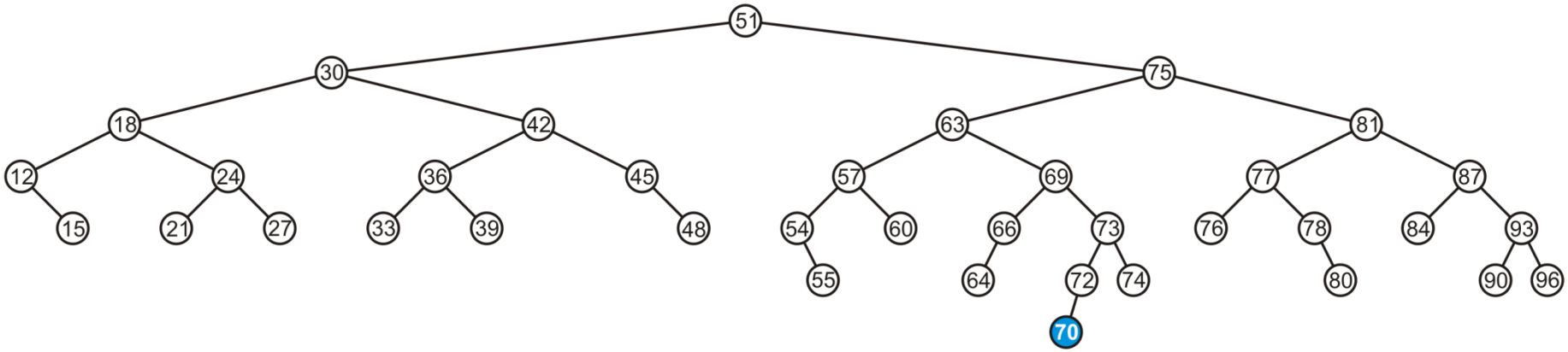
Insertion

The tree is now balanced



Insertion

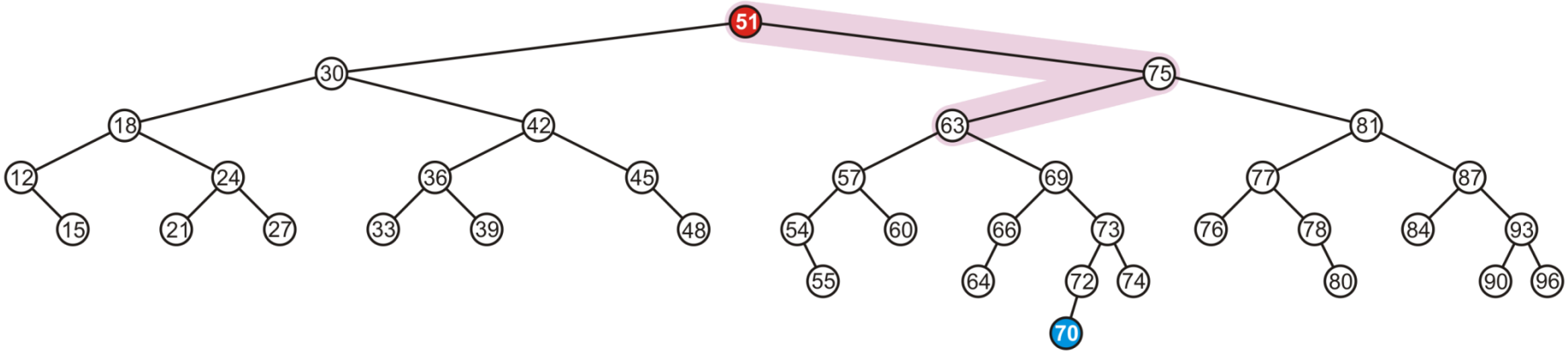
Insert 70



Insertion

The root node is now imbalanced

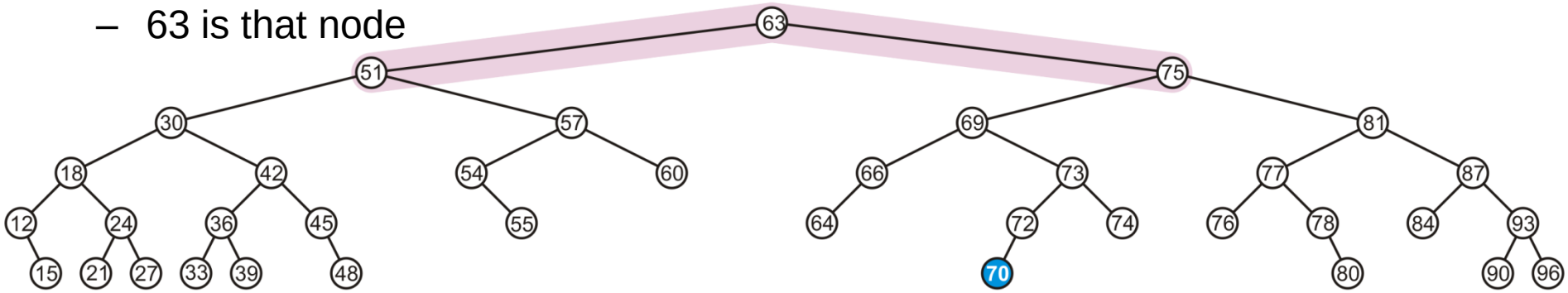
- A right-left imbalance
- Promote the intermediate node to the root



Insertion

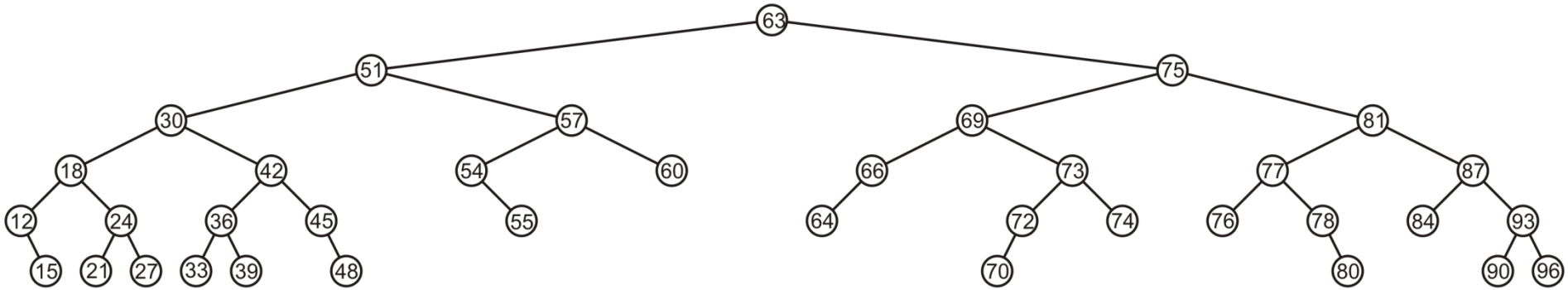
The root node is imbalanced

- A right-left imbalance
- Promote the intermediate node to the root
- 63 is that node



Insertion

The result is AVL balanced



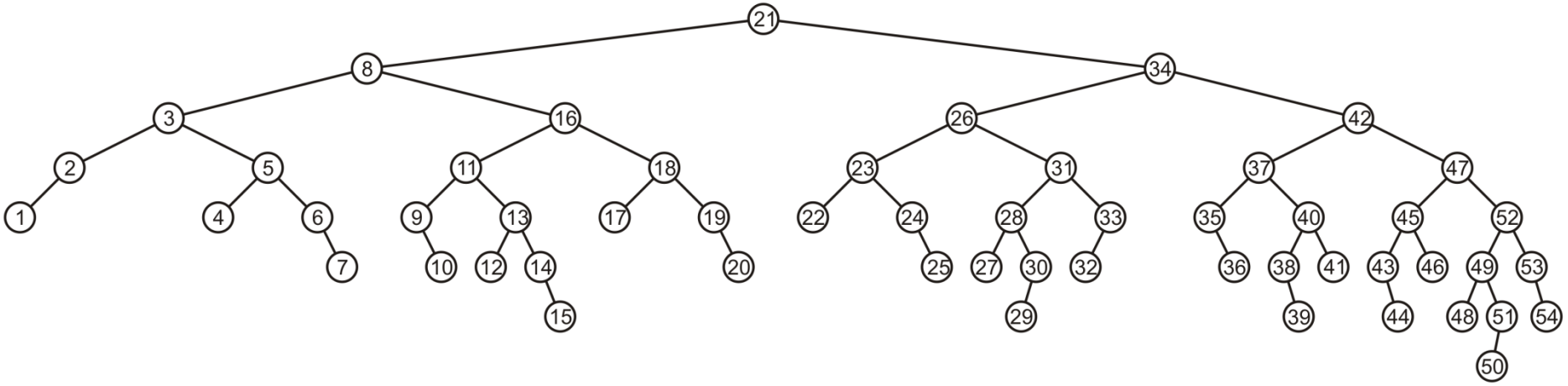
Deletion

Removing a node from an AVL tree may cause more than one nodes to be imbalance

- Node can be deleted using BST Delete function
- Node height of all nodes along the path from the parent node to the root must be updated
- Node from the parent node to the root must be recursively checked for the balance property
- If not balance, fix it.
- Check direction that cause imbalancing
  - Left-Left direction: use SingleRotationFromLeft
  - Right-Right direction: use SingleRotationFromRight
  - Left-Right direction: use DoubleRotationFromLeft
  - Right-Left direction: use DoubleRotationFromRight

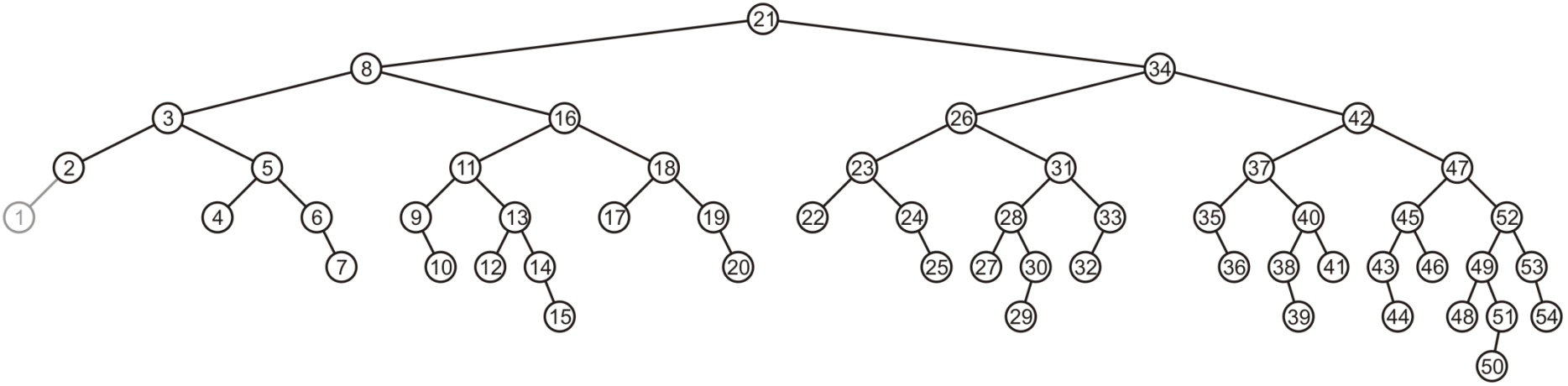
Deletion

Consider the following AVL tree



Deletion

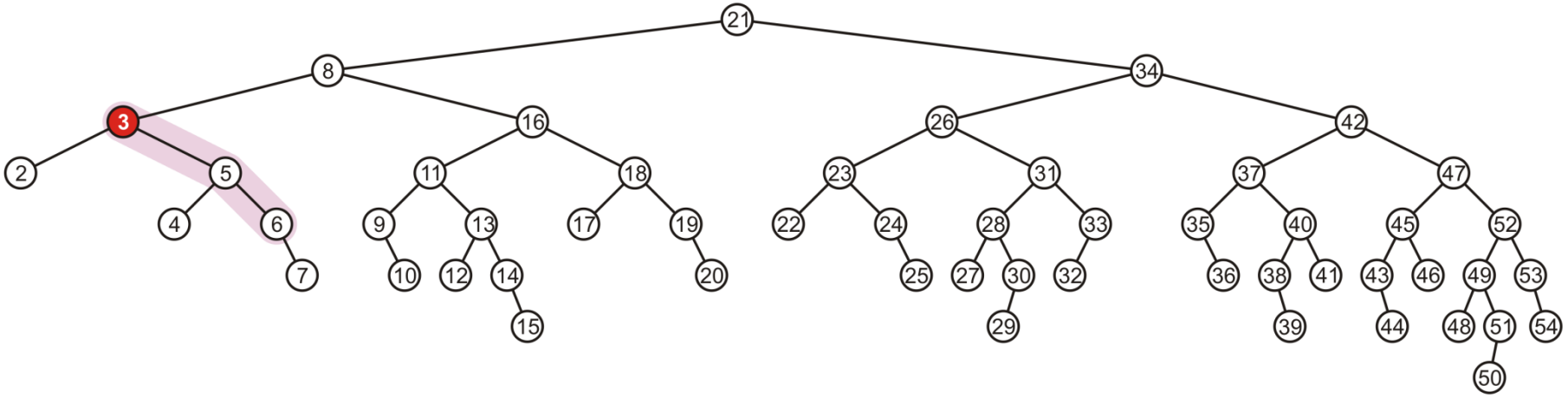
Suppose we erase the front node: 1



Deletion

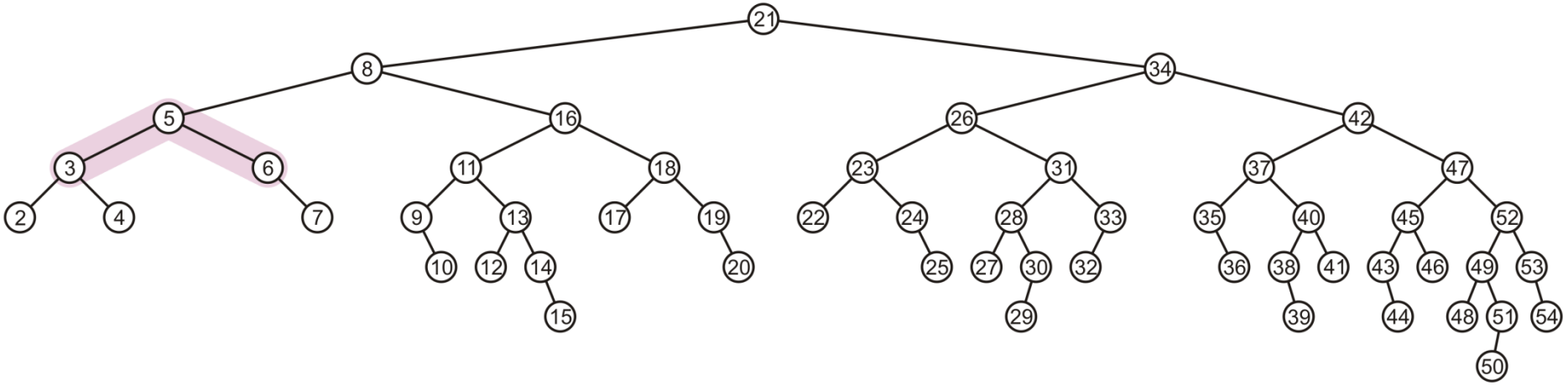
While its previous parent, 2, is not unbalanced, its grandparent 3 is

- The imbalance is in the right-right subtree



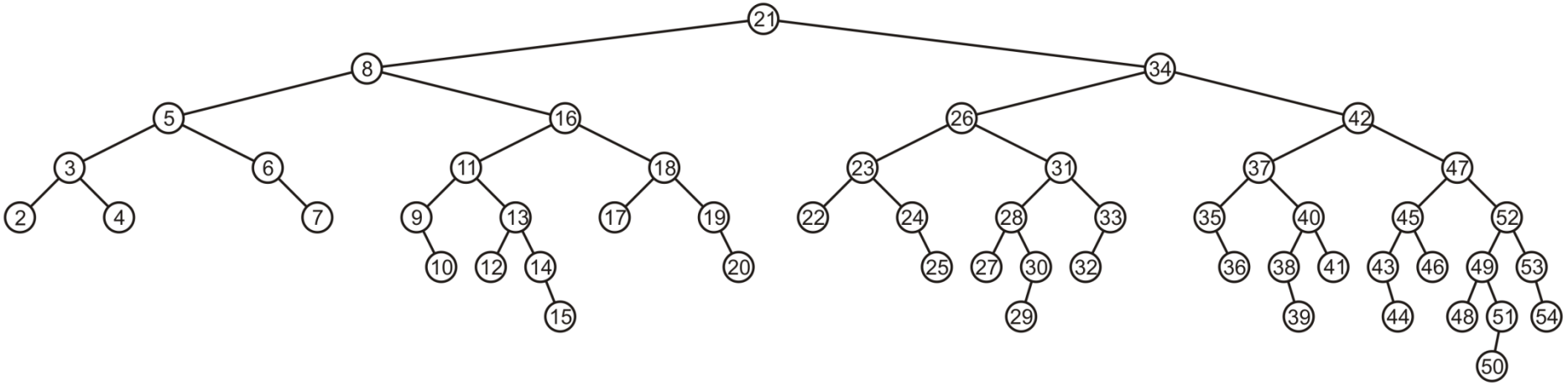
Deletion

We can correct this with a simple balance



Deletion

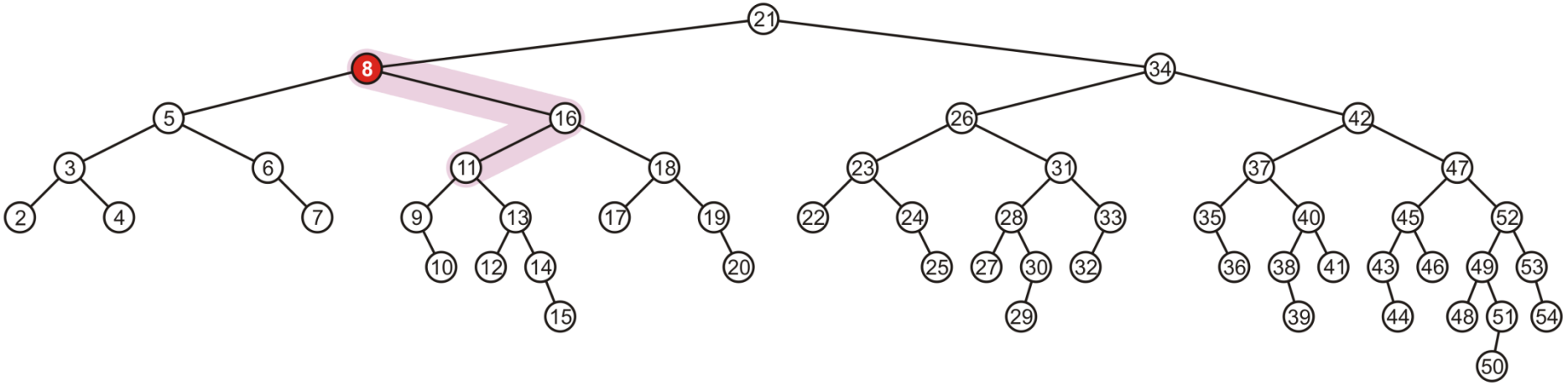
The node of that subtree, 5, is now balanced



Deletion

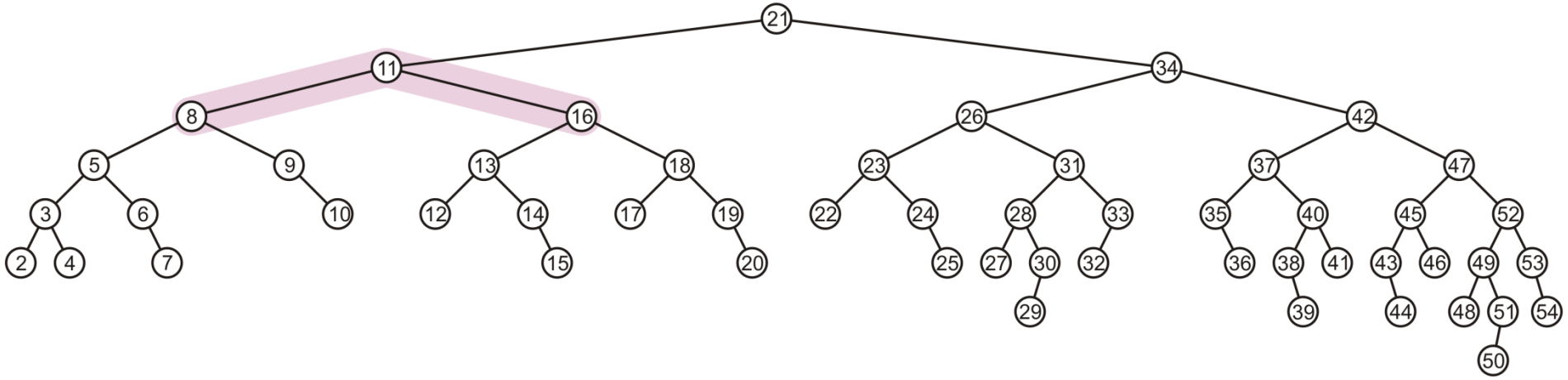
Recursing to the root, however, 8 is also unbalanced

- This is a right-left imbalance



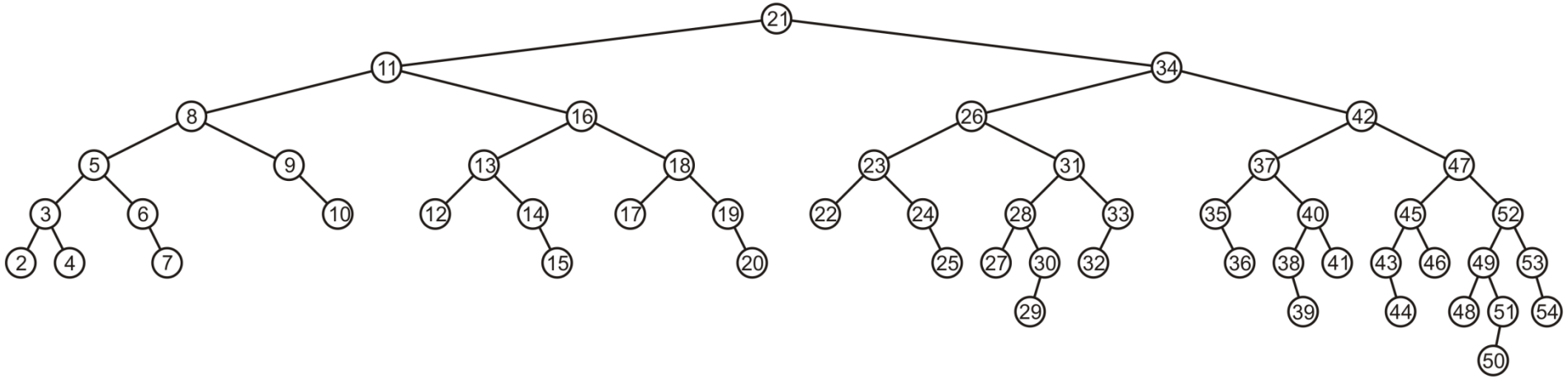
Deletion

Promoting 11 to the root corrects the imbalance



Deletion

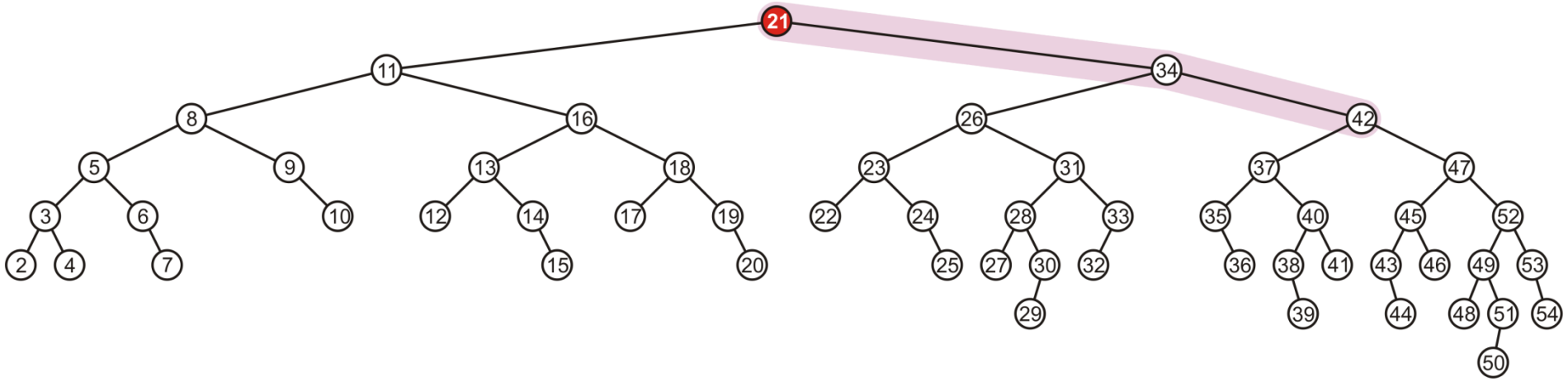
At this point, the node 11 is balanced



Deletion

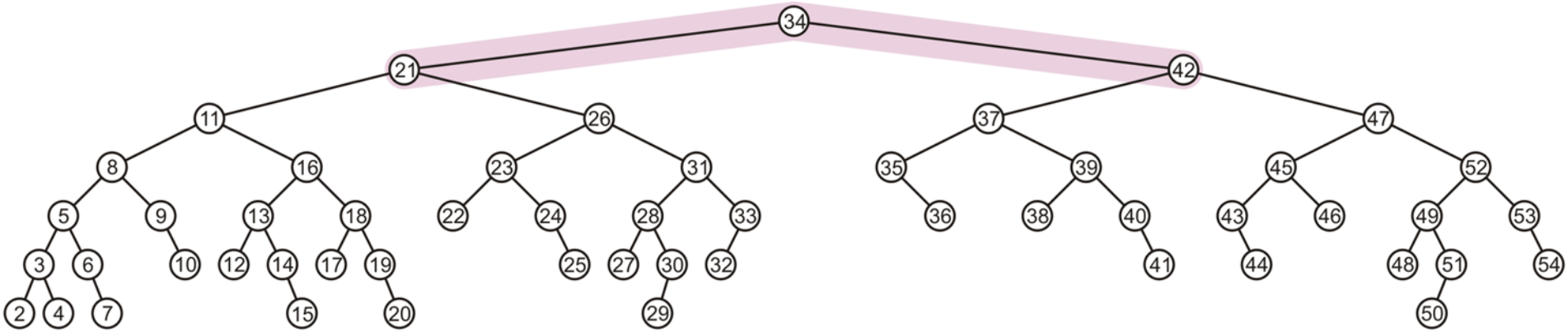
Still, the root node is unbalanced

- This is a right-right imbalance



Deletion

Again, a simple balance fixes the imbalance



Deletion

The resulting tree is now AVL balanced