AVL Trees

COL 106

Amit Kumar

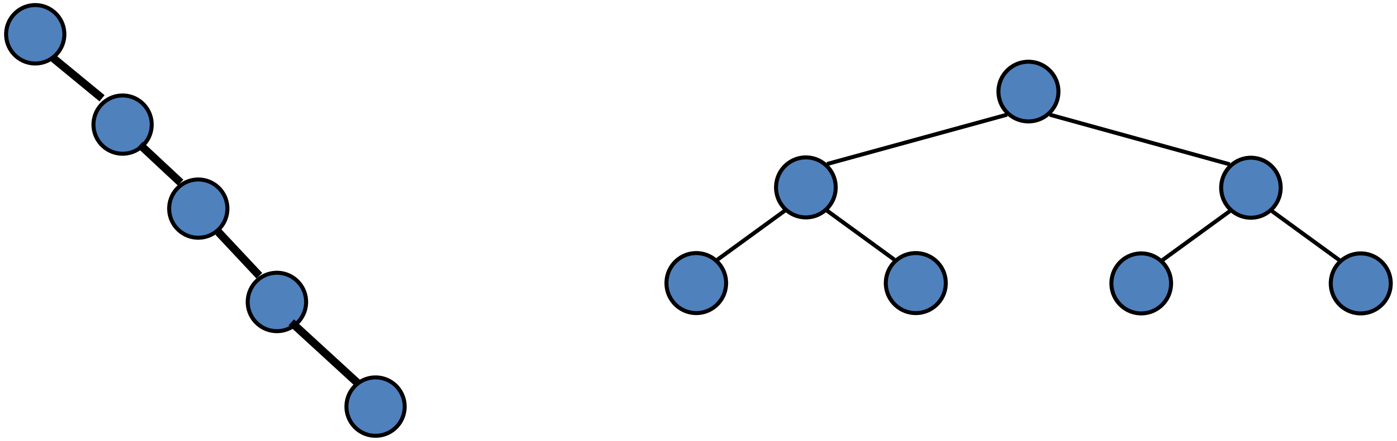
Shweta Agrawal

Slide Courtesy : Douglas Wilhelm Harder, MMath, UWaterloo

`dwharder@alumni.uwaterloo.ca`

Problems with BST

- Running time of Insert and Delete depend upon the height of the BST.
- But the height of a BST on n nodes can be close to n.



Balance BST

Requirement:

- Define and maintain a *balance* condition to ensure $\Theta(\ln(n))$ height
- What are natural balance conditions ?

AVL Trees

- – Named after Adelson-Velskii and Landis

Notion of balance in AVL trees?

Balance is defined by comparing the height of the two sub-trees

Recall:

- – An empty tree has height -1
- – A tree with a single node has height 0

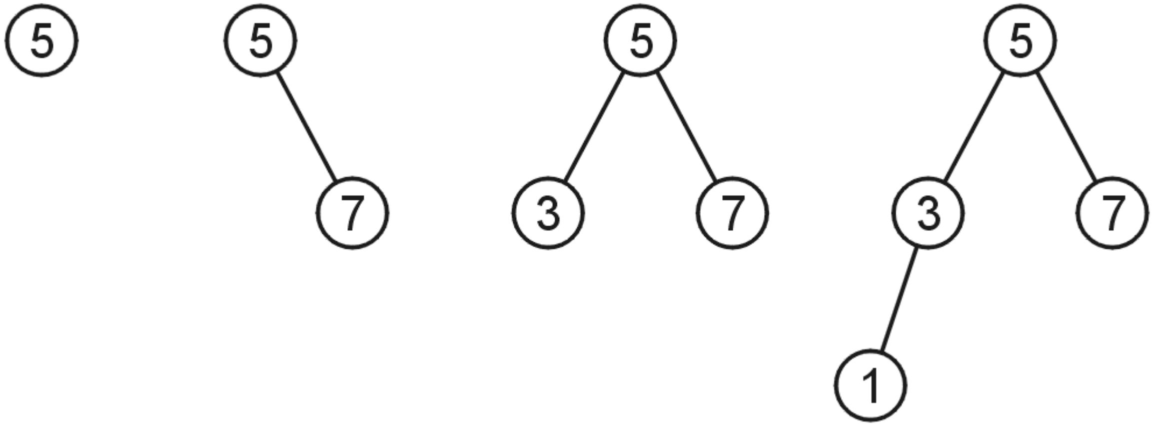
AVL Trees

A binary search tree is said to be AVL balanced 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

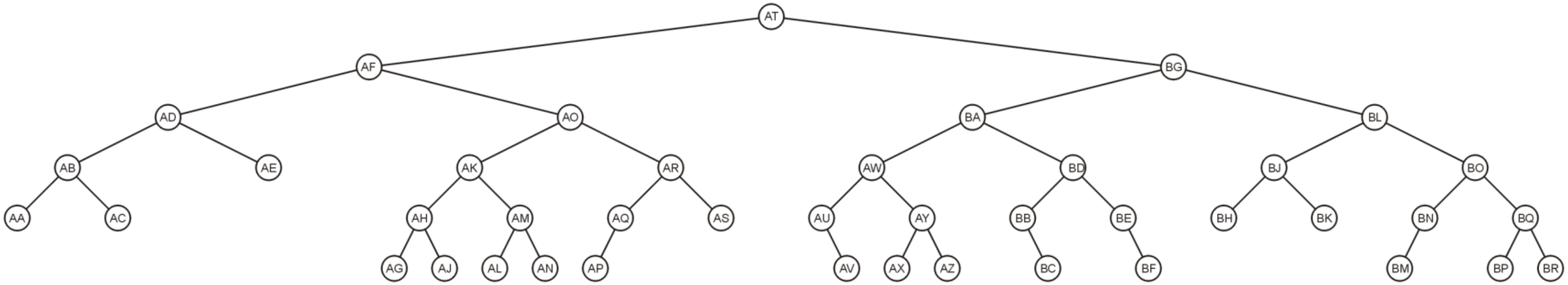
AVL Trees

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



AVL Trees

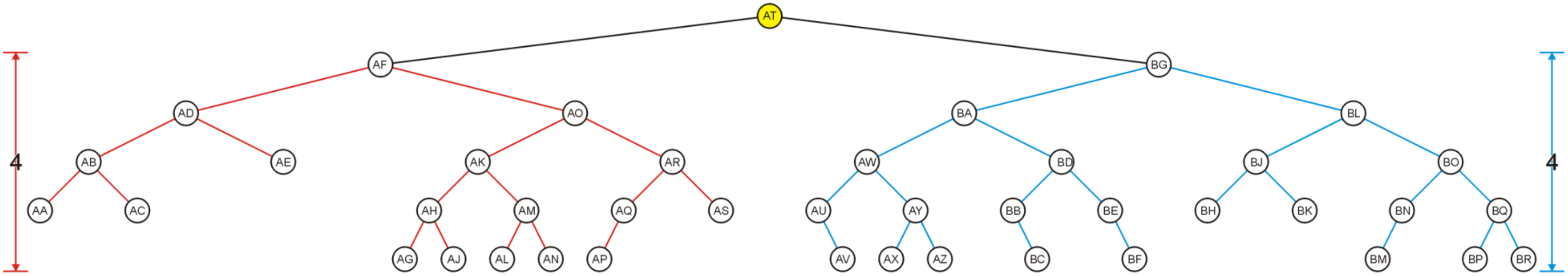
Here is a larger AVL tree (42 nodes):



AVL Trees

The root node is AVL-balanced:

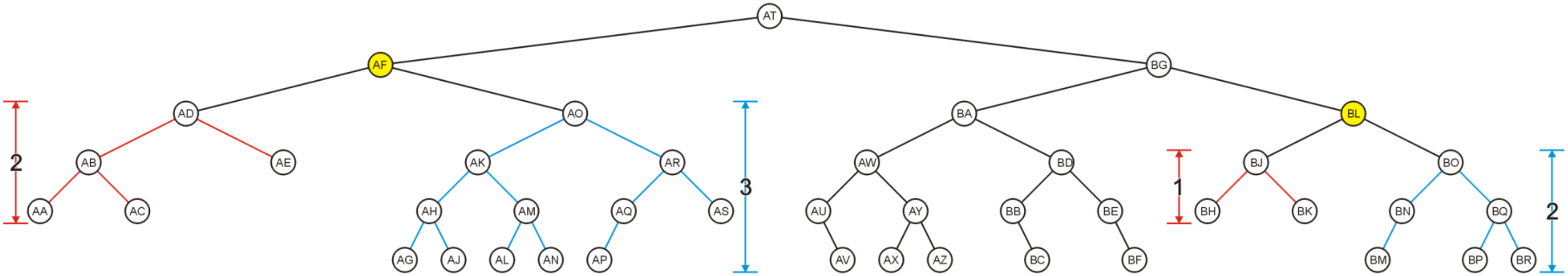
- Both sub-trees are of height 4:



AVL Trees

All other nodes are AVL balanced

- The sub-trees differ in height by at most one



Height of an AVL Tree

By the definition of complete trees, any complete binary search tree is an AVL tree

Thus an upper bound on the number of nodes in an AVL tree of height h

a perfect binary tree with $2^{h+1} - 1$ nodes

– What is a lower bound?

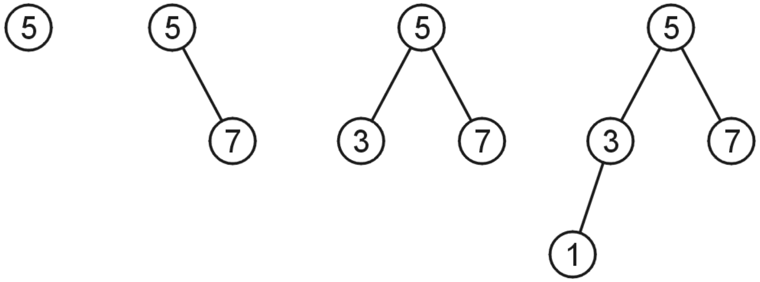
Height of an AVL Tree

H(n): Height of an AVL tree on n nodes

Not well defined!

H(n): worst possible height of an AVL tree on n nodes

Want to show H(n) is O(log n).



Height of an AVL Tree

- Write a recurrence for H(n):

H(n) <= H(n/2) + 2

Looks like the recurrence for binary search.

Implies H(n) = O(log n)

Maintaining Balance

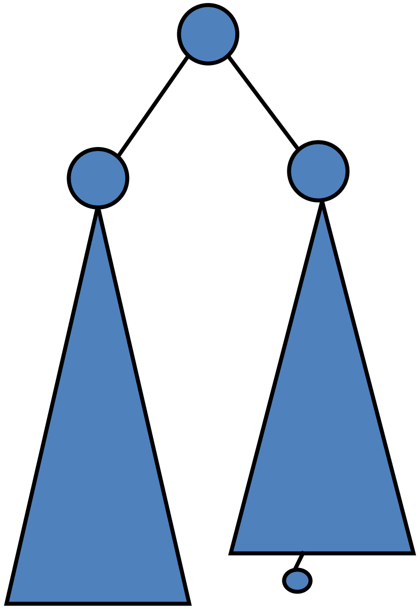
To maintain AVL balance, observe that:

- 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

Insertion in an AVL Tree

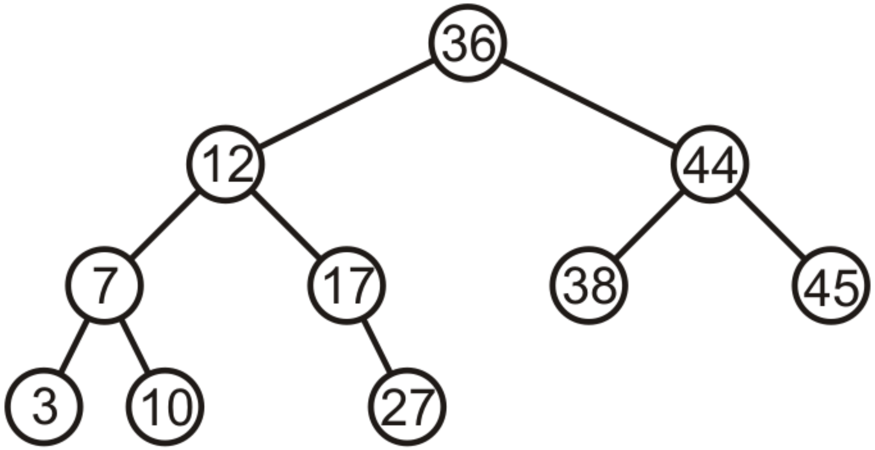
- Insert as in a BST.
- If height condition maintained at each node, we are done!

How much time does it take to check this condition ?



Maintaining Balance

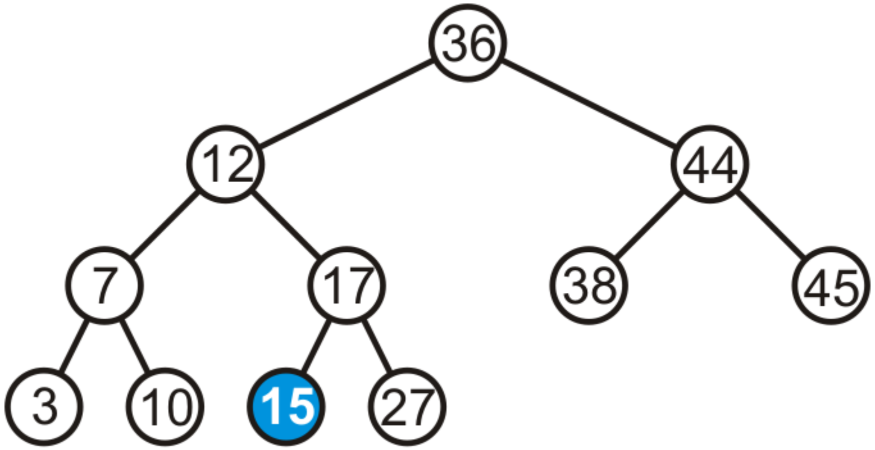
Consider this AVL tree



Maintaining Balance

Consider inserting 15 into this tree

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

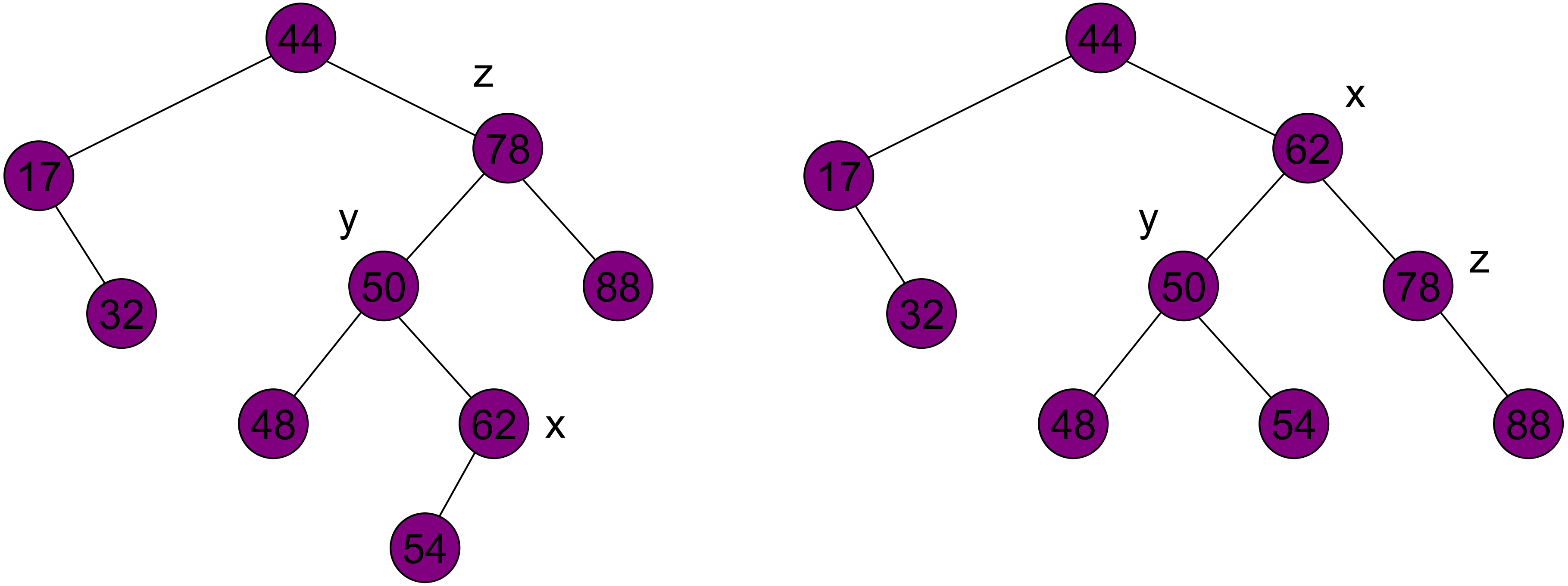


Insertion

- Inserting a node, v, into an AVL tree changes the heights of some of the nodes in T.
- The only nodes whose heights can increase are the ancestors of node v.
- If insertion causes T to become unbalanced, then some ancestor of v would have a height-imbalance.
- We travel up the tree from v until we find the first node x such that its grandparent z is unbalanced.
- Let y be the parent of node x.
- Rearrange by making the middle element the parent of the other two.

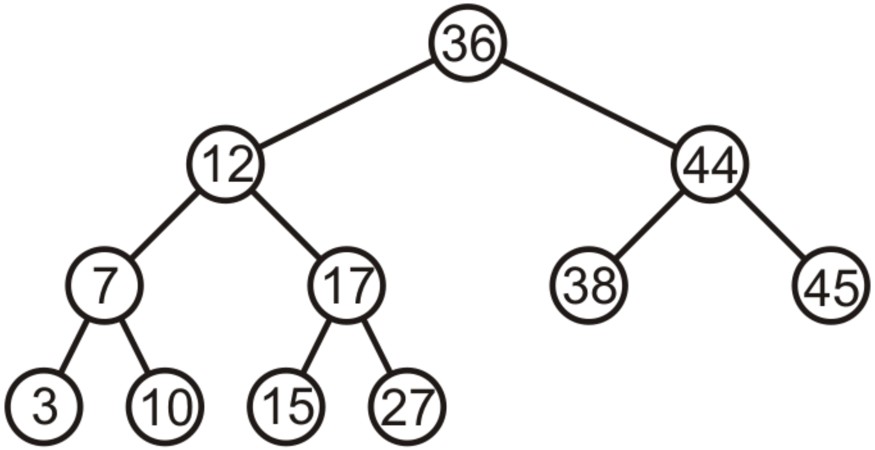
Insertion (2)

To rebalance the subtree rooted at z, we must perform a *rotation*.



Maintaining Balance

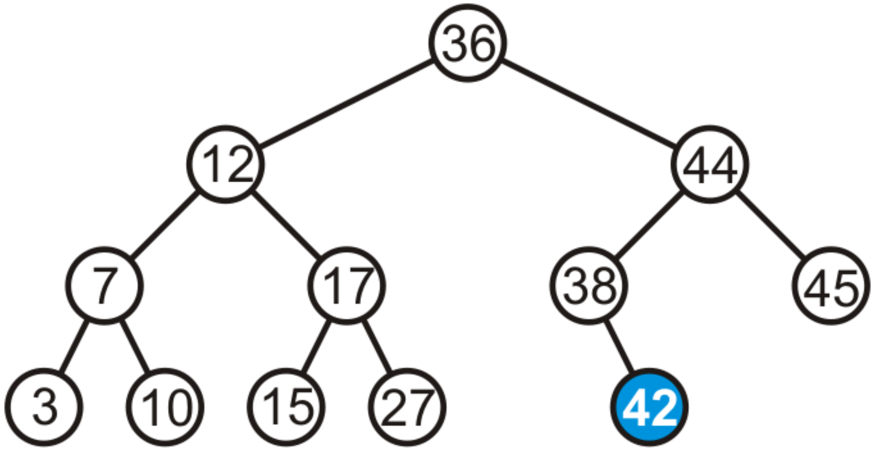
The tree remains balanced



Maintaining Balance

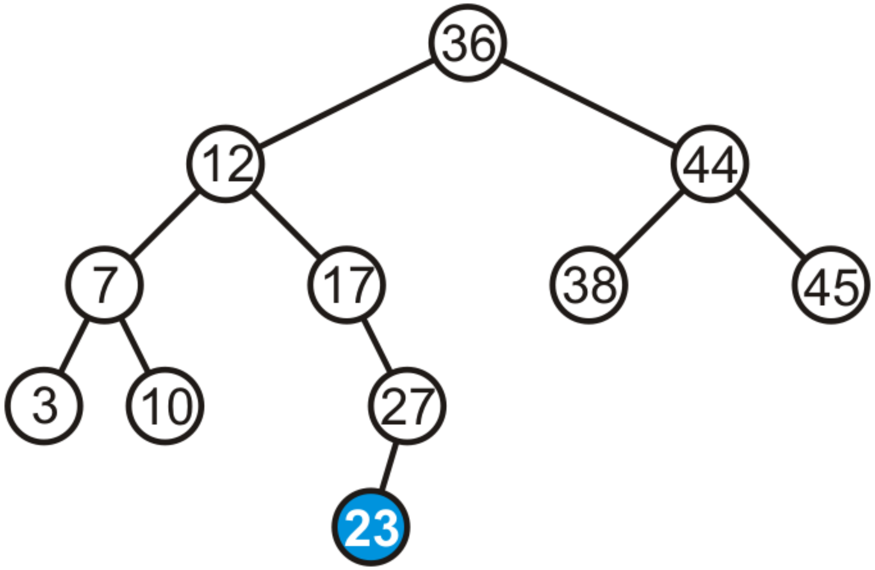
Consider inserting 42 into this tree

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



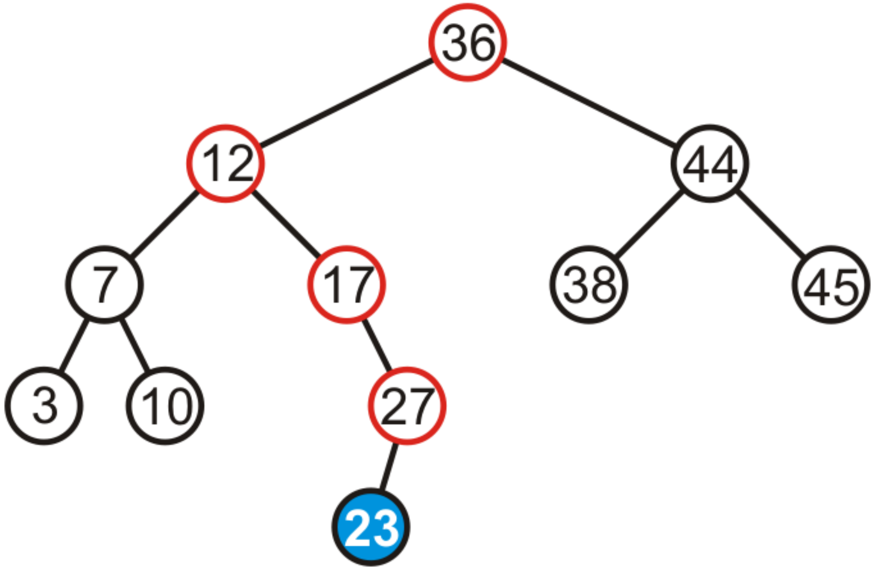
Maintaining Balance

Suppose we insert 23 into our initial tree



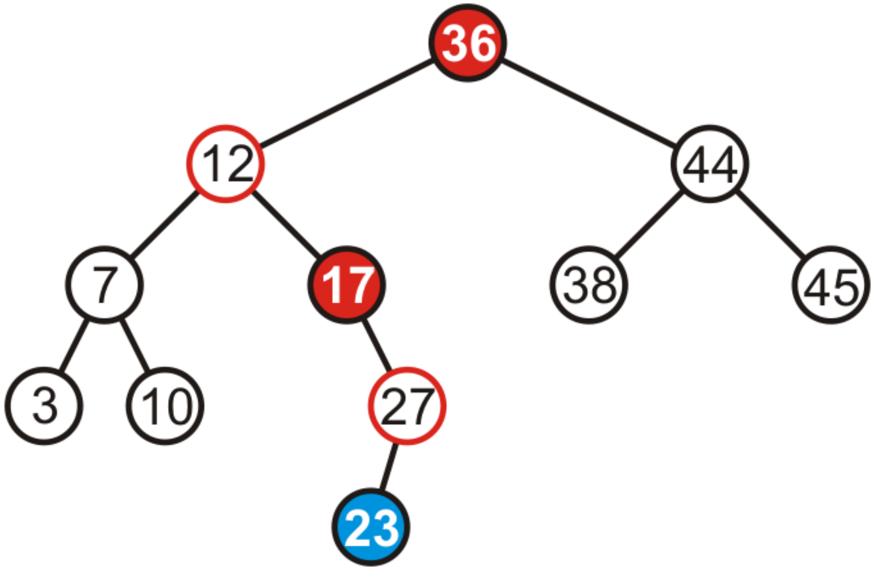
Maintaining Balance

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



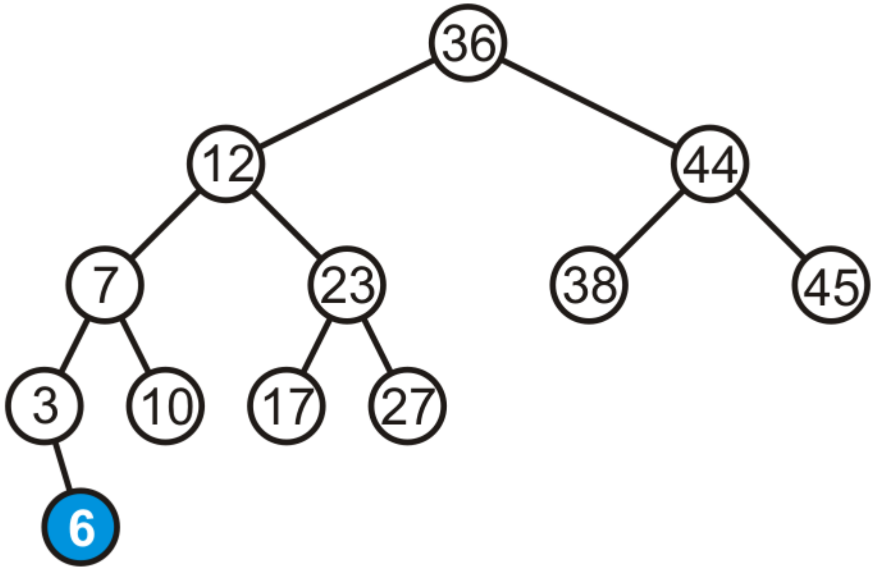
Maintaining Balance

However, only two of the nodes are unbalanced: 17 and 36



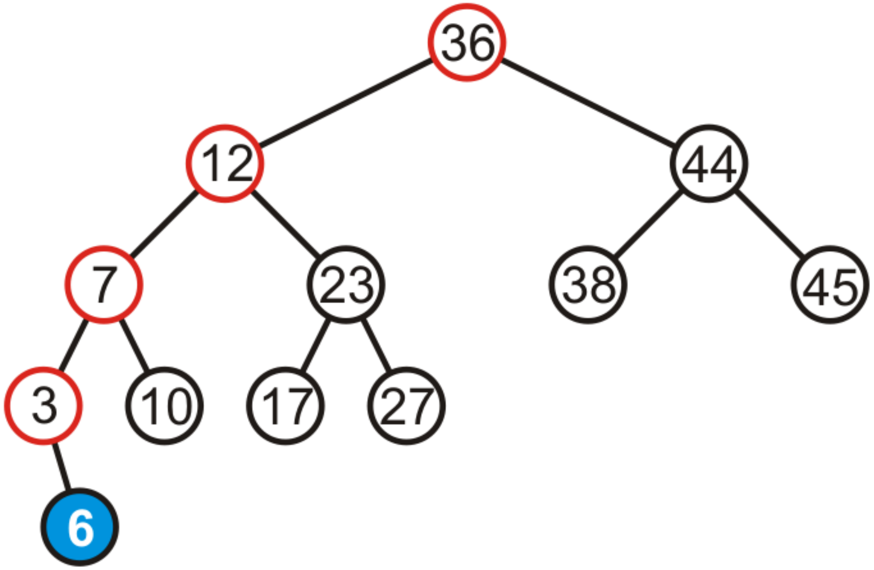
Maintaining Balance

Consider adding 6:



Maintaining Balance

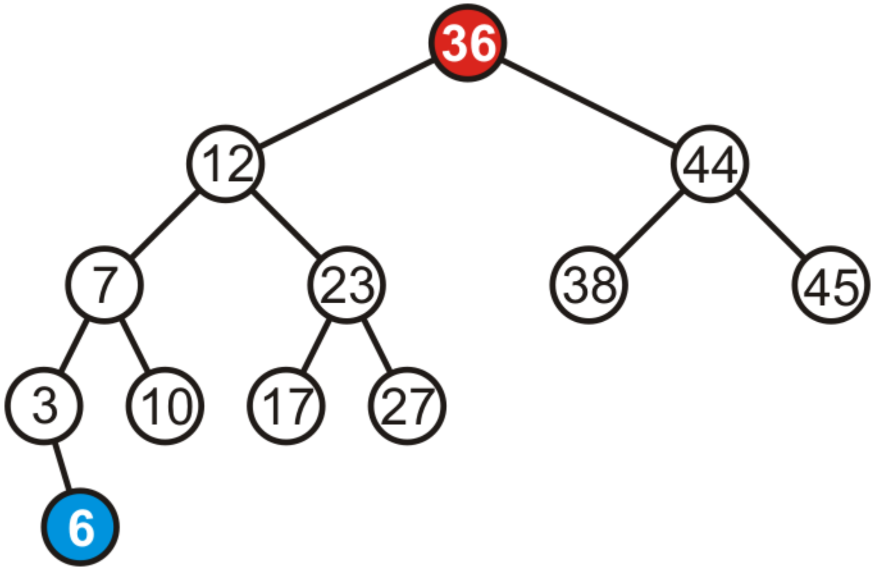
The height of each of the trees in the path back to the root are increased by one



Maintaining Balance

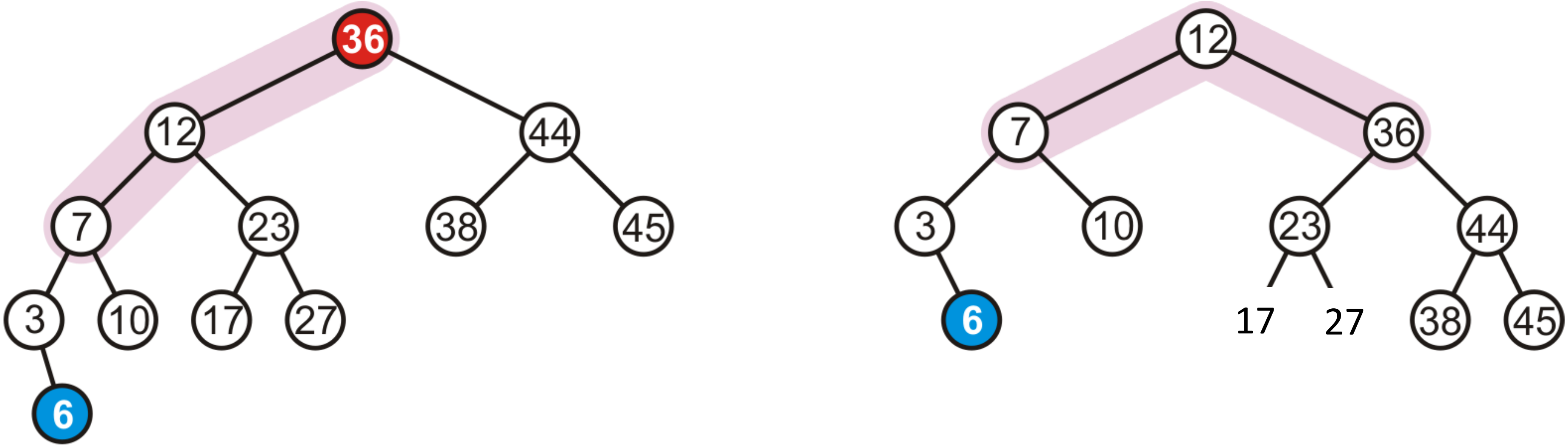
The height of each of the trees in the path back to the root are increased by one

- However, only the root node is now unbalanced



Maintaining Balance

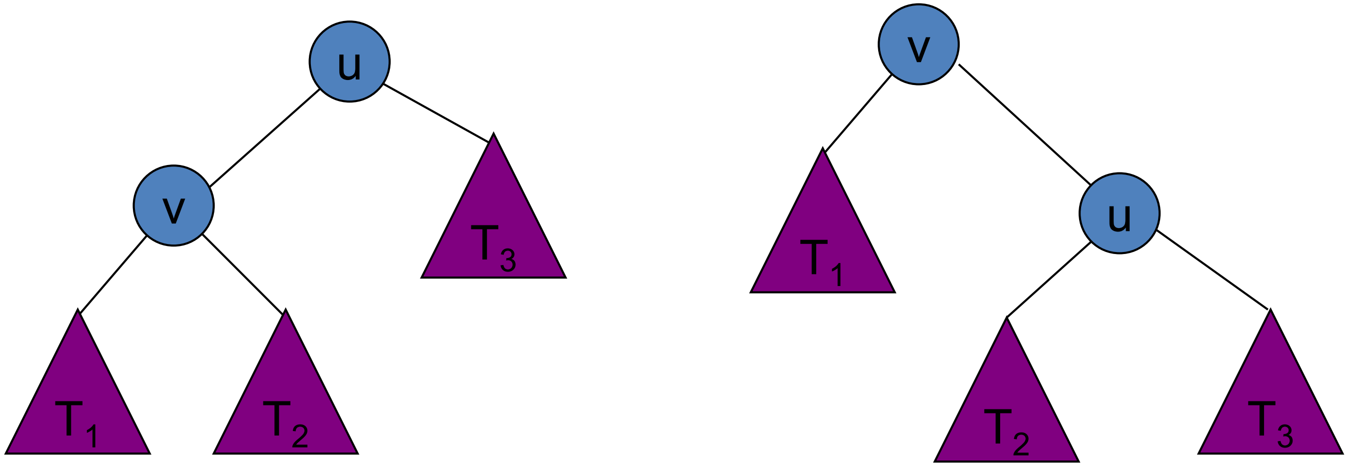
We may fix this by rotating the root to the right



Note: the right subtree of 12 became the left subtree of 36

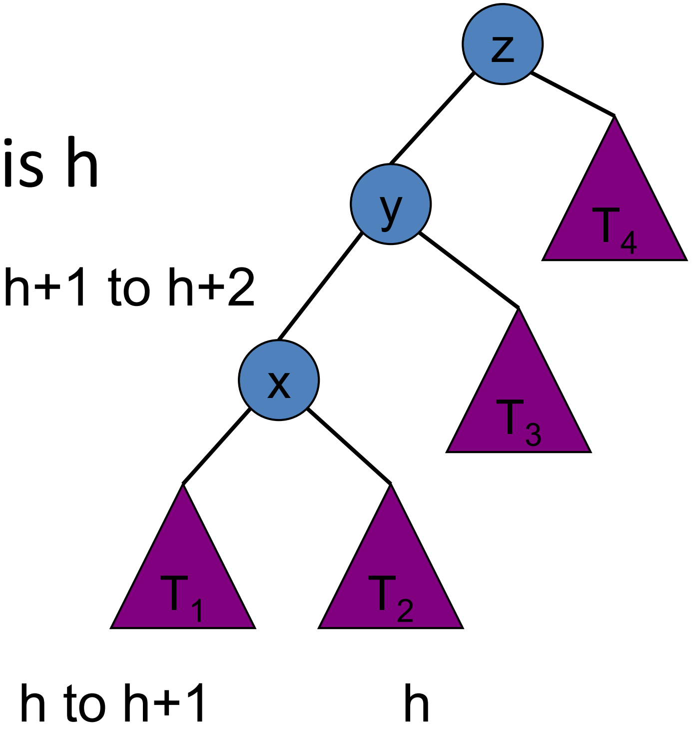
Rotations

- Rotation is a way of locally reorganizing a BST.
- Let u,v be two nodes such that u=parent(v)
- $\text{Keys}(T_1) < \text{key}(v) < \text{keys}(T_2) < \text{key}(u) < \text{keys}(T_3)$

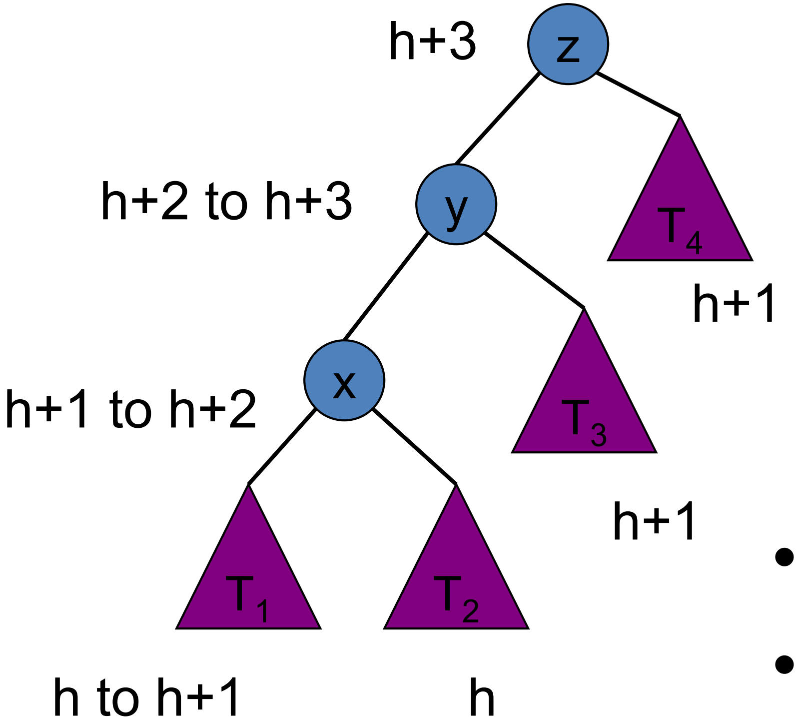


Insertion

- Insertion happens in subtree T_1.
- $ht(T_1)$ increases from h to h+1.
- Since x remains balanced $ht(T_2)$ is h or h+1 or h+2.
  - If $ht(T_2)$=h+2 then x is originally unbalanced
  - If $ht(T_2)$=h+1 then ht(x) does not increase.
  - Hence $ht(T_2)$=h.
- So ht(x) increases from h+1 to h+2.

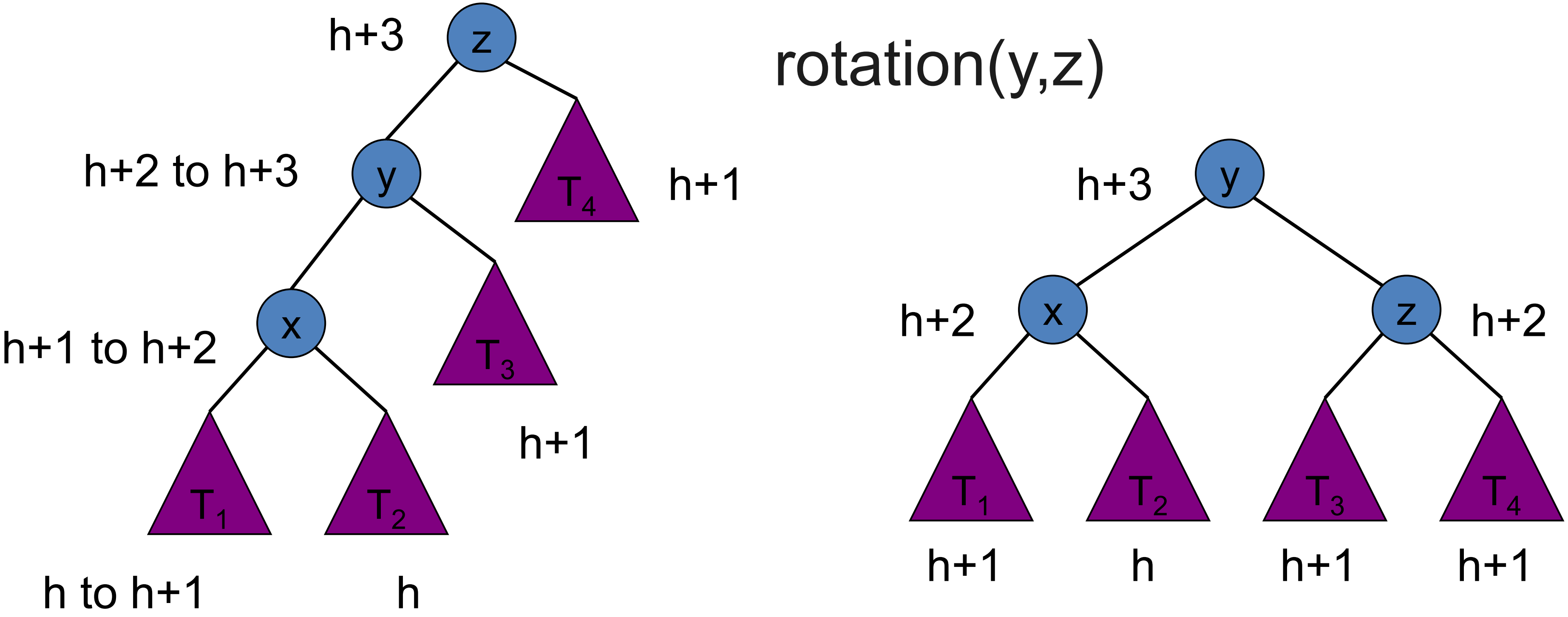


Insertion(2)



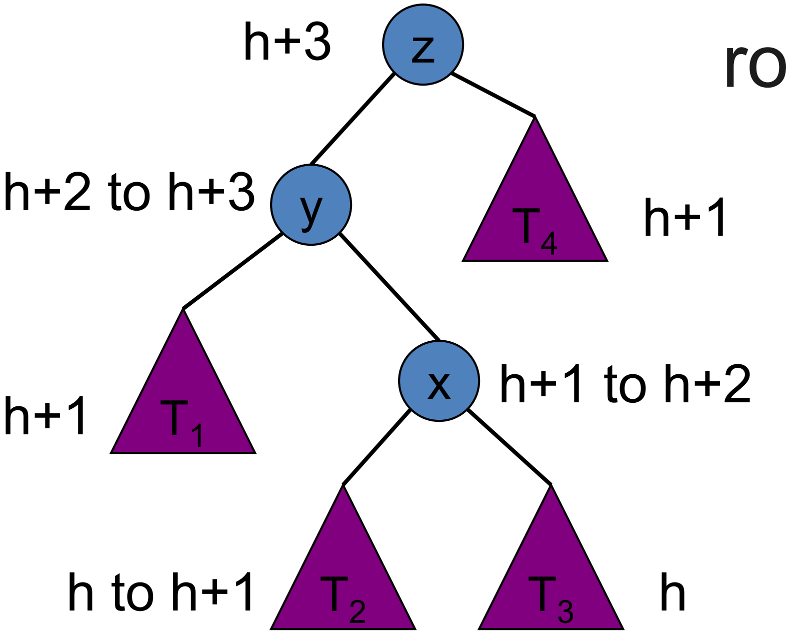
- Since y remains balanced, ht(T_3) is h+1 or h+2 or h+3.
  - If ht(T_3)=h+3 then y is originally unbalanced.
  - If ht(T_3)=h+2 then ht(y) does not increase.
  - So ht(T_3)=h+1.
- So ht(y) inc. from h+2 to h+3.
- Since z was balanced ht(T_4) is h+1 or h+2 or h+3.
- z is now unbalanced and so ht(T_4)=h+1.

Single rotation

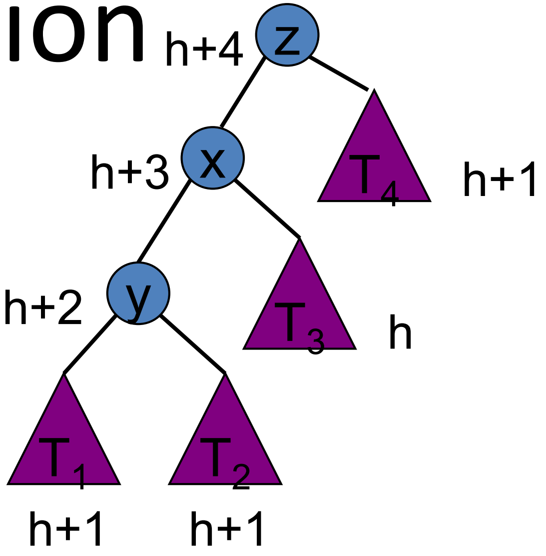


The height of the subtree remains the same after rotation. Hence no further rotations required

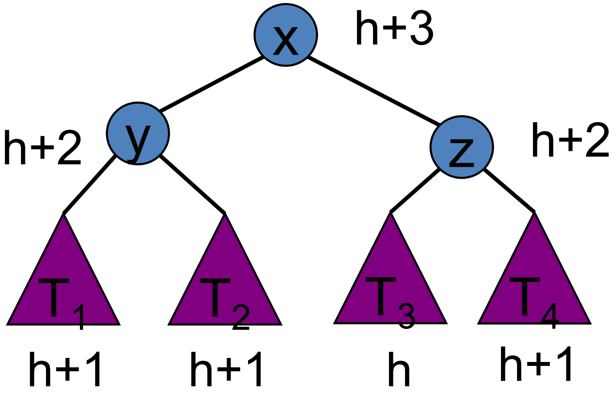
Double rotation



rotation(x,y)



rotation(x,z)

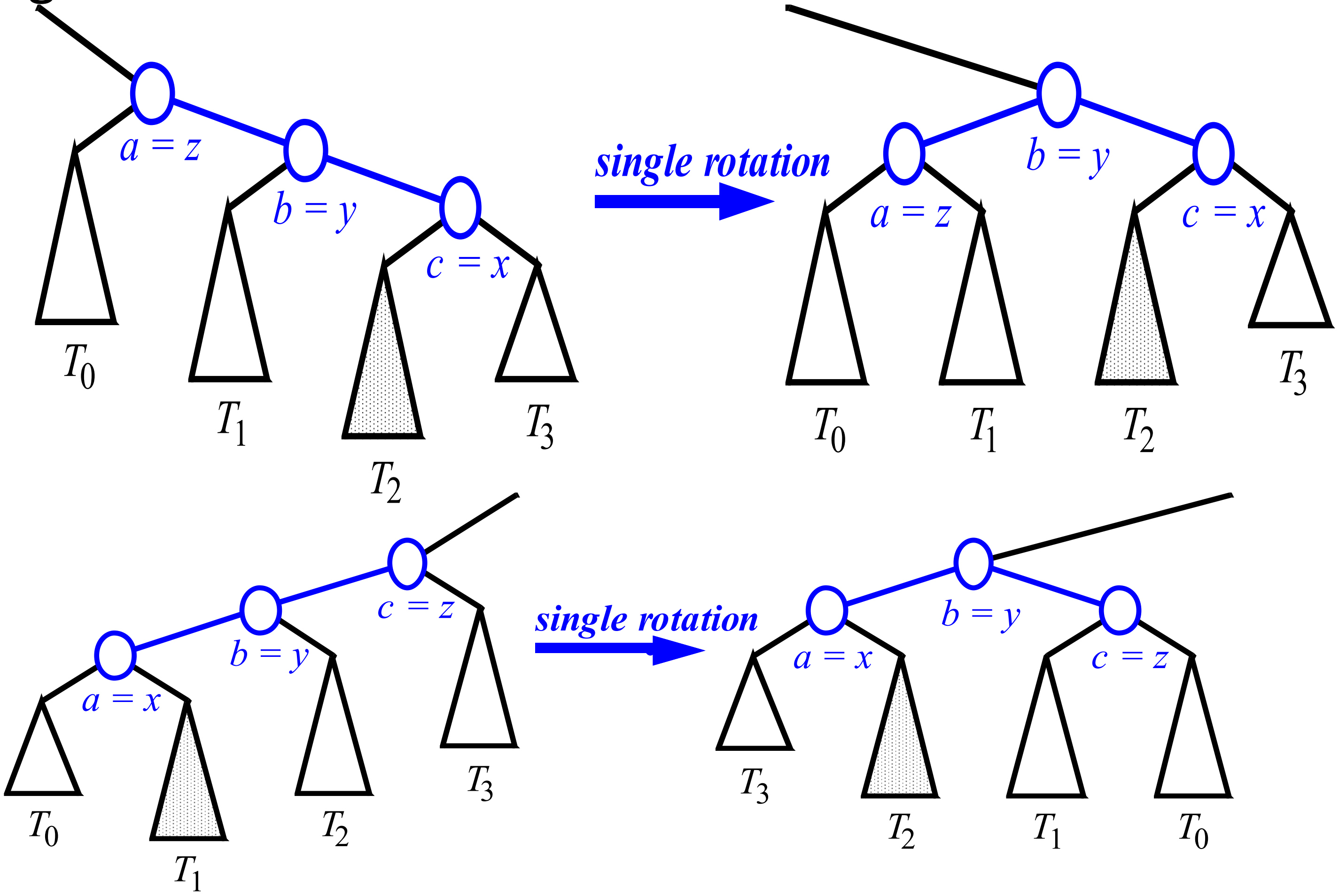


Final tree has same height as original tree. Hence we need not go further up the tree.

Restructuring

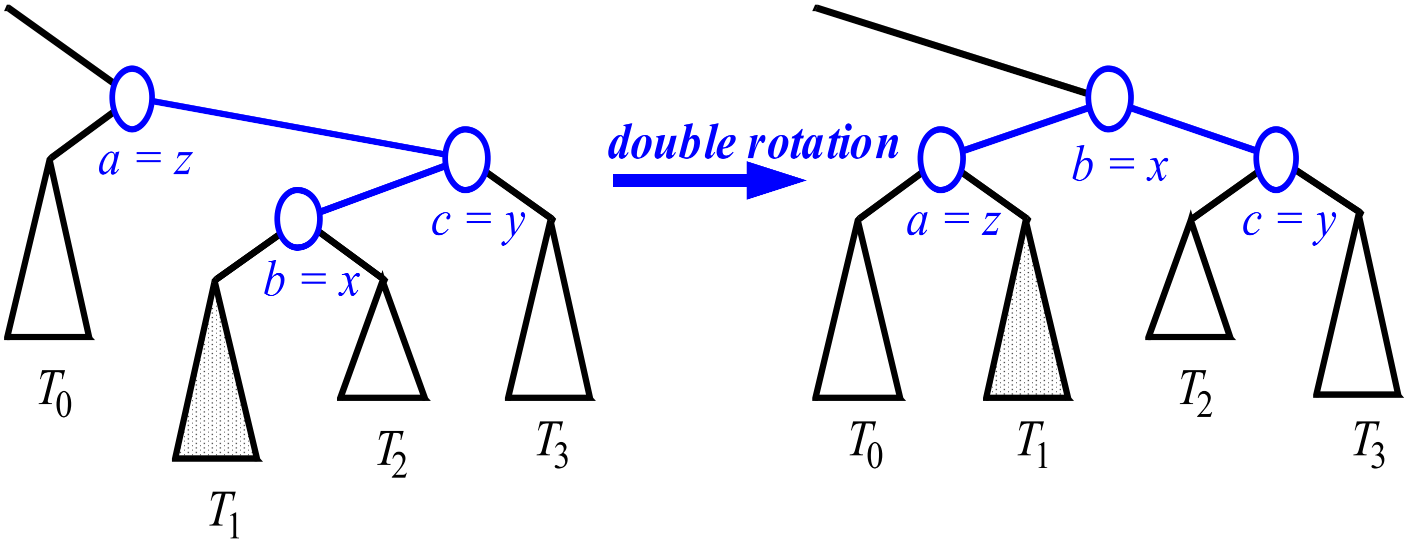
- The four ways to rotate nodes in an AVL tree, graphically represented

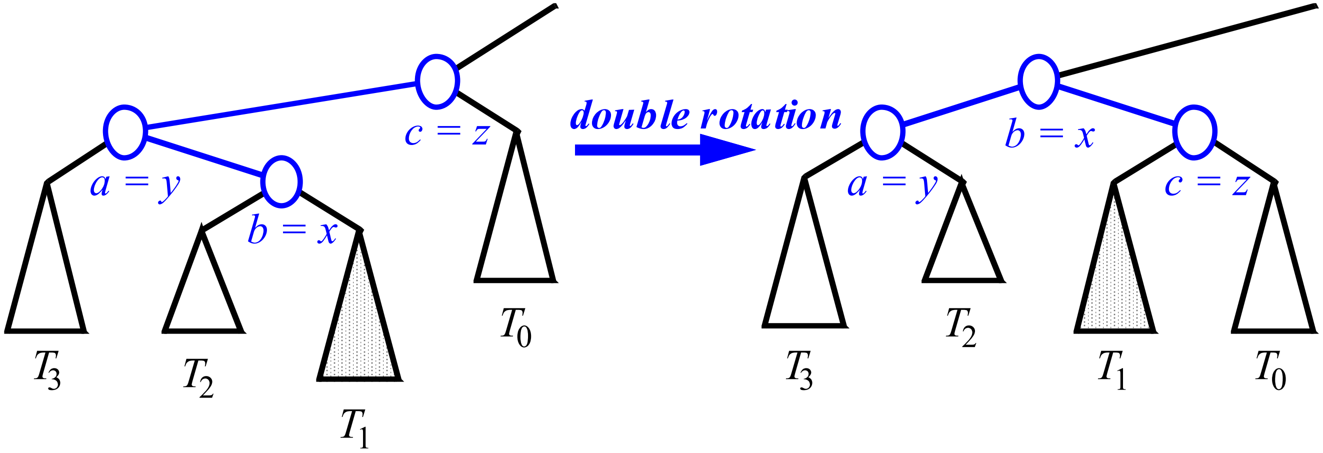
  -Single Rotations:



Restructuring (contd.)

- double rotations:

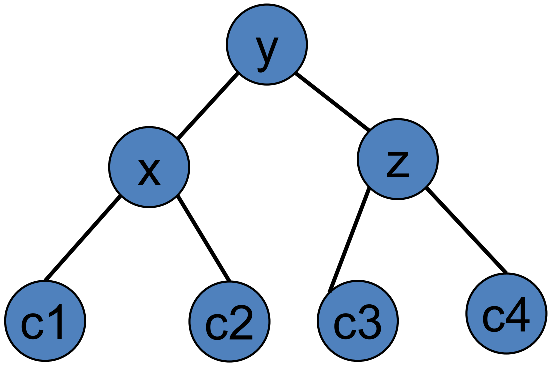




Implementation Trick

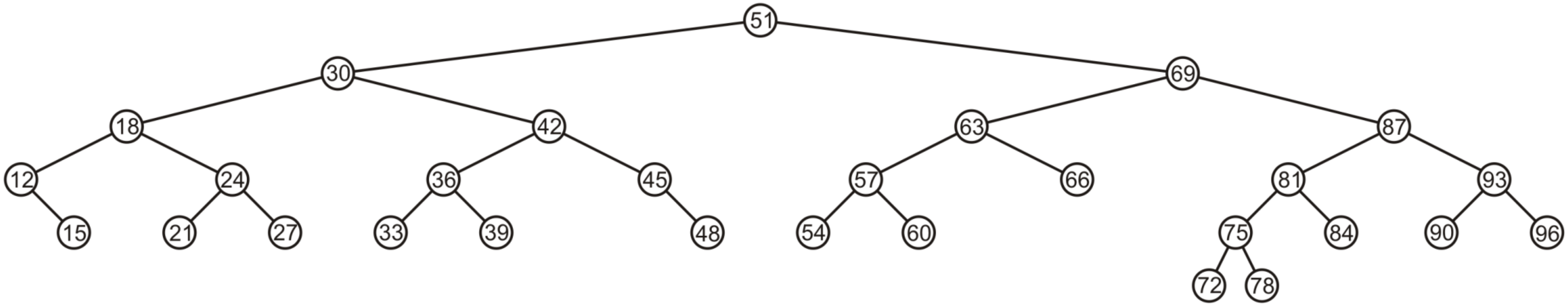
Arrange x,y,z and their 4 children (which could be NULL) in increasing order.

c1	x	c2	y	c3	z	c4



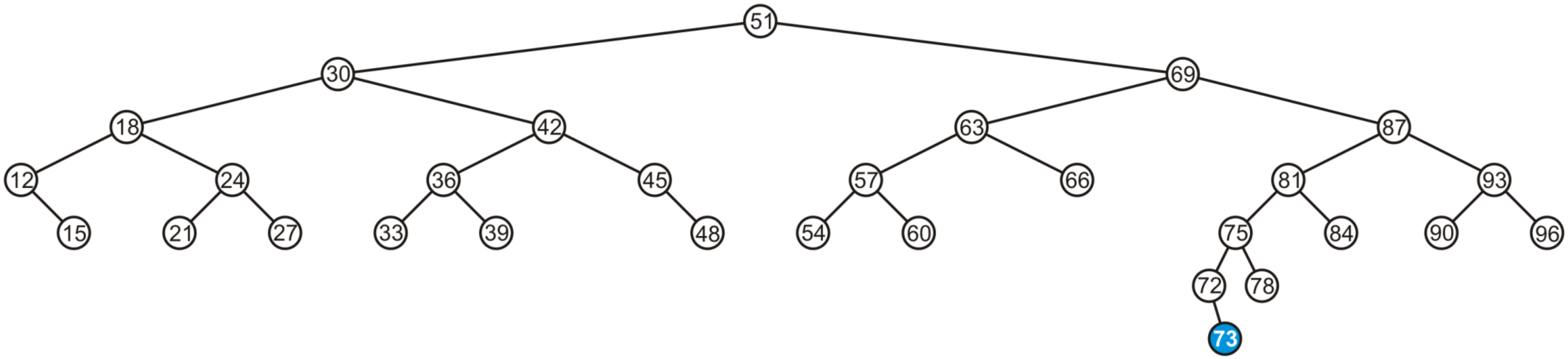
More examples : Insertion

Consider this AVL tree



Insertion

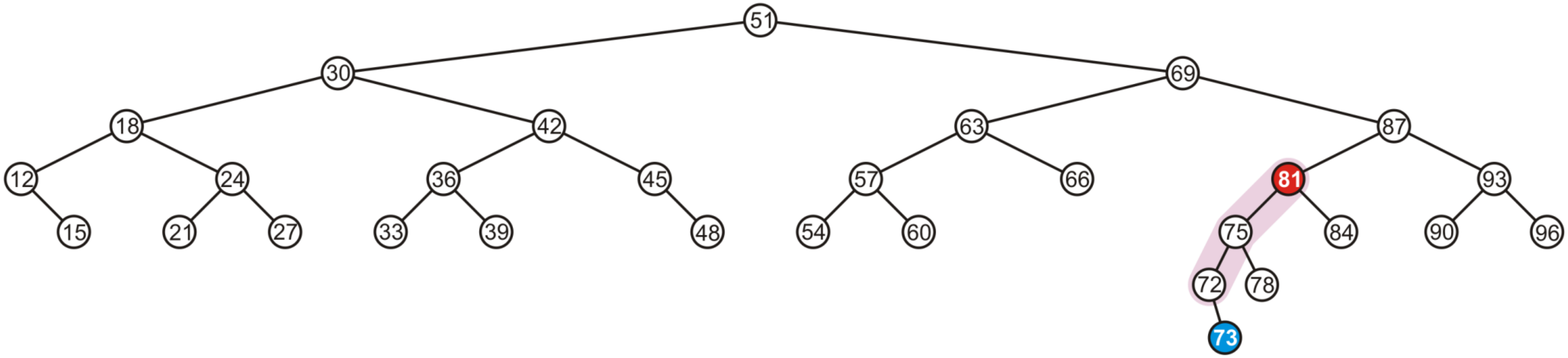
Insert 73



Insertion

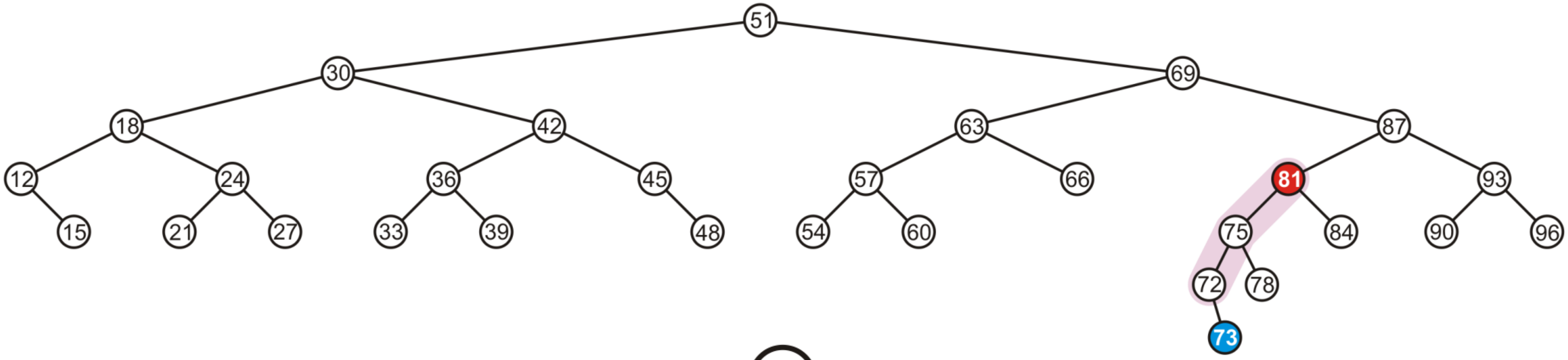
The node 81 is unbalanced

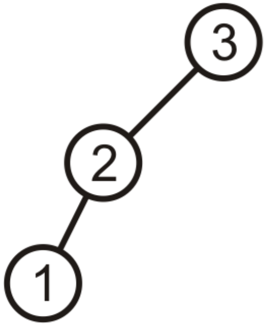
- A left-left imbalance



Insertion

- The node 81 is unbalanced
  - A left-left imbalance

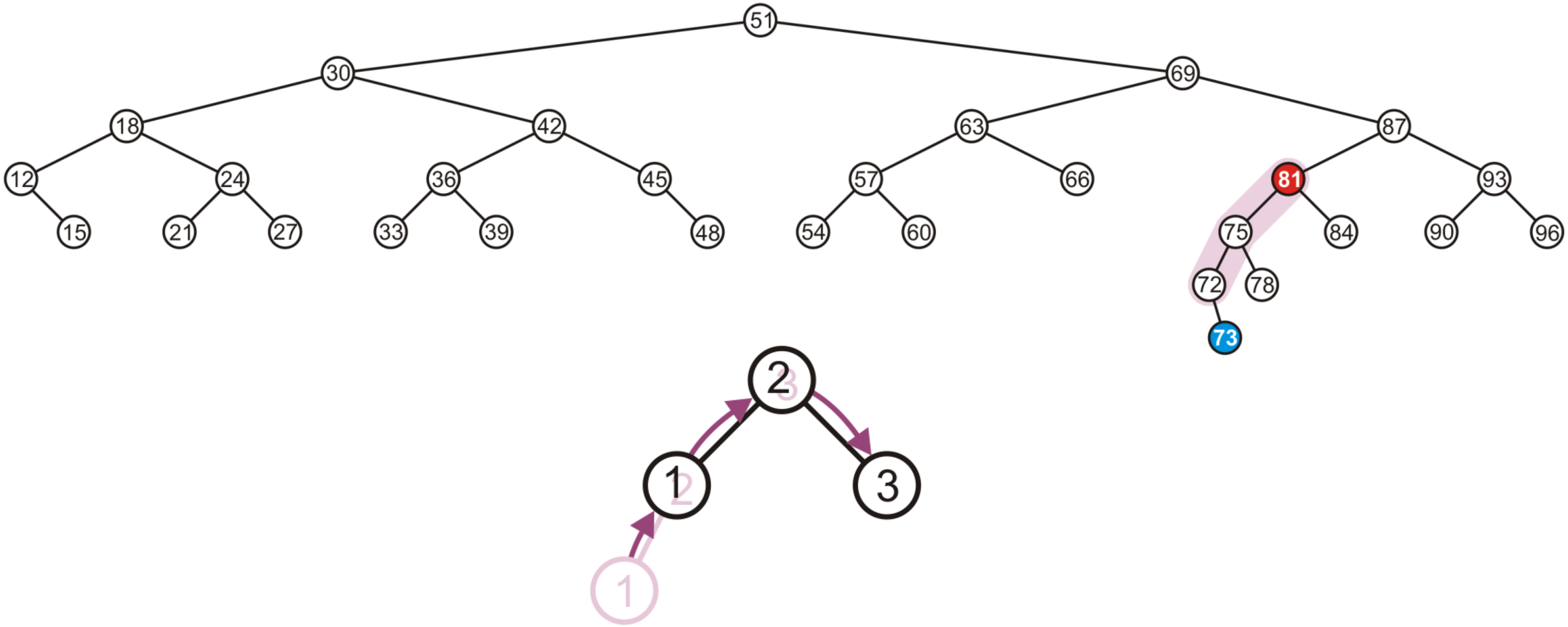




Insertion

The node 81 is unbalanced

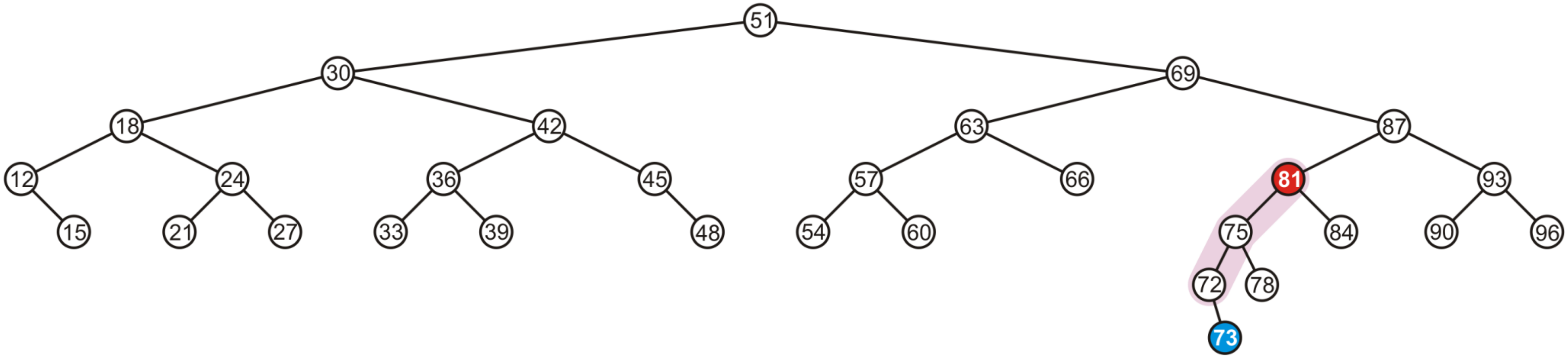
- A left-left imbalance



Insertion

The node 81 is unbalanced

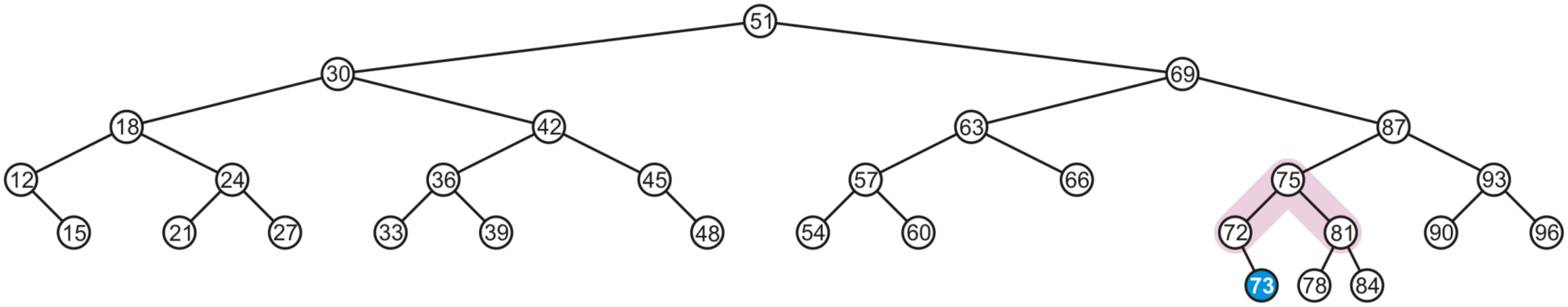
- A left-left imbalance
- Promote the intermediate node to the imbalanced node
- 75 is that node



Insertion

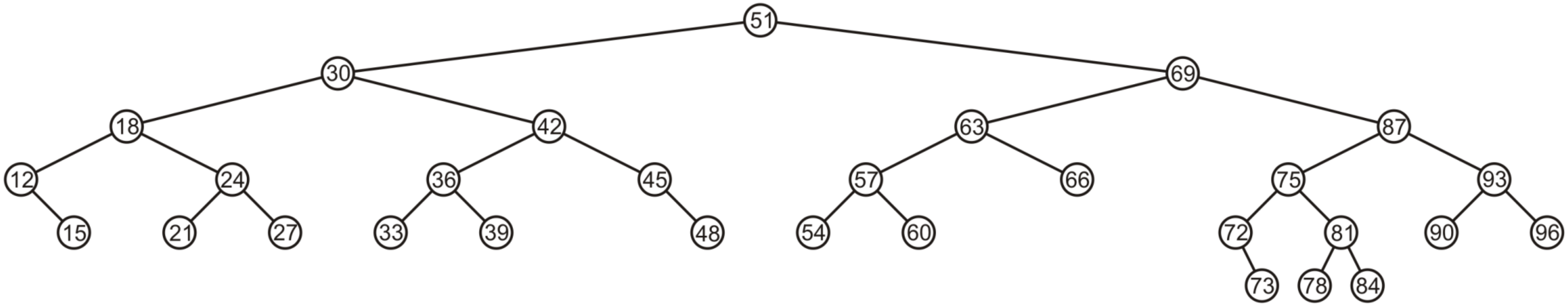
The node 81 is unbalanced

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



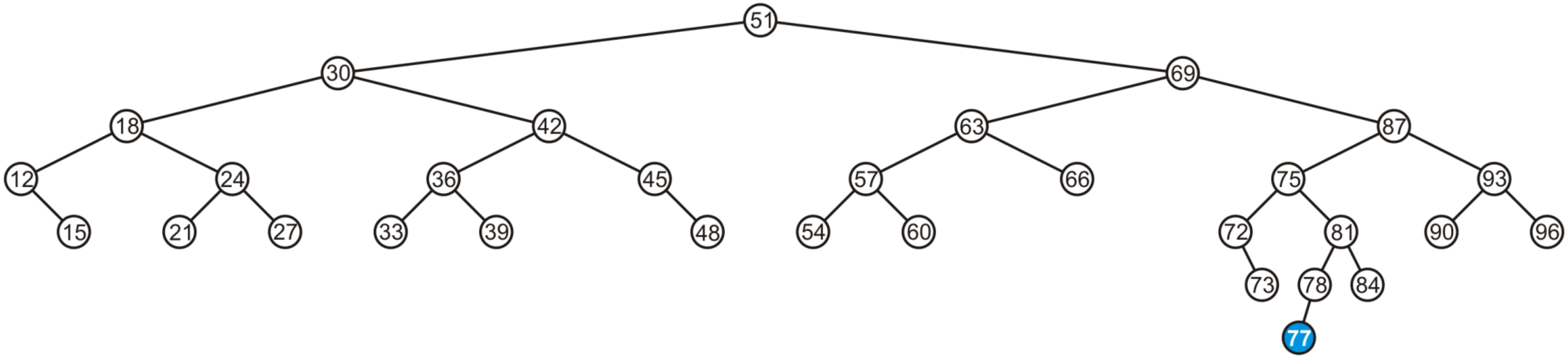
Insertion

The tree is AVL balanced



Insertion

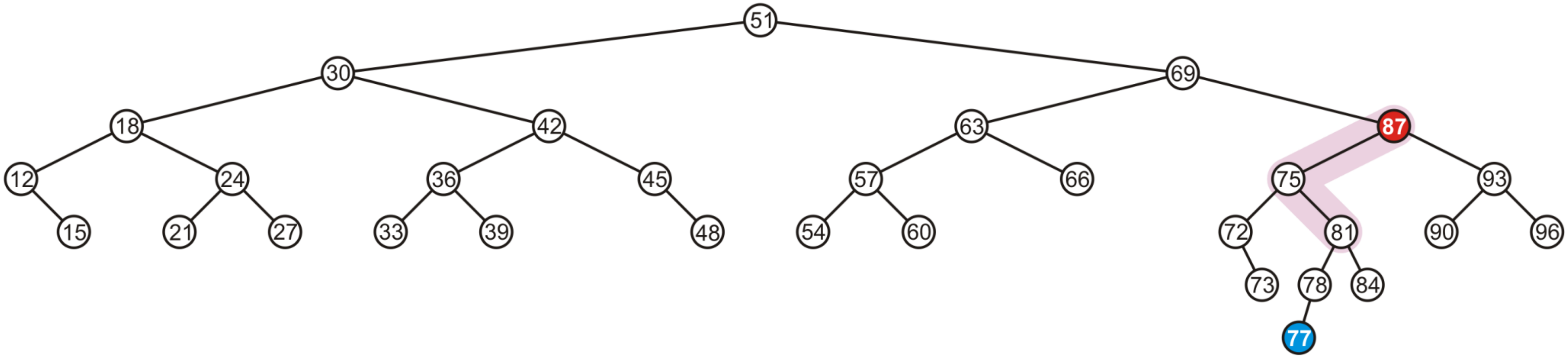
Insert 77



Insertion

The node 87 is unbalanced

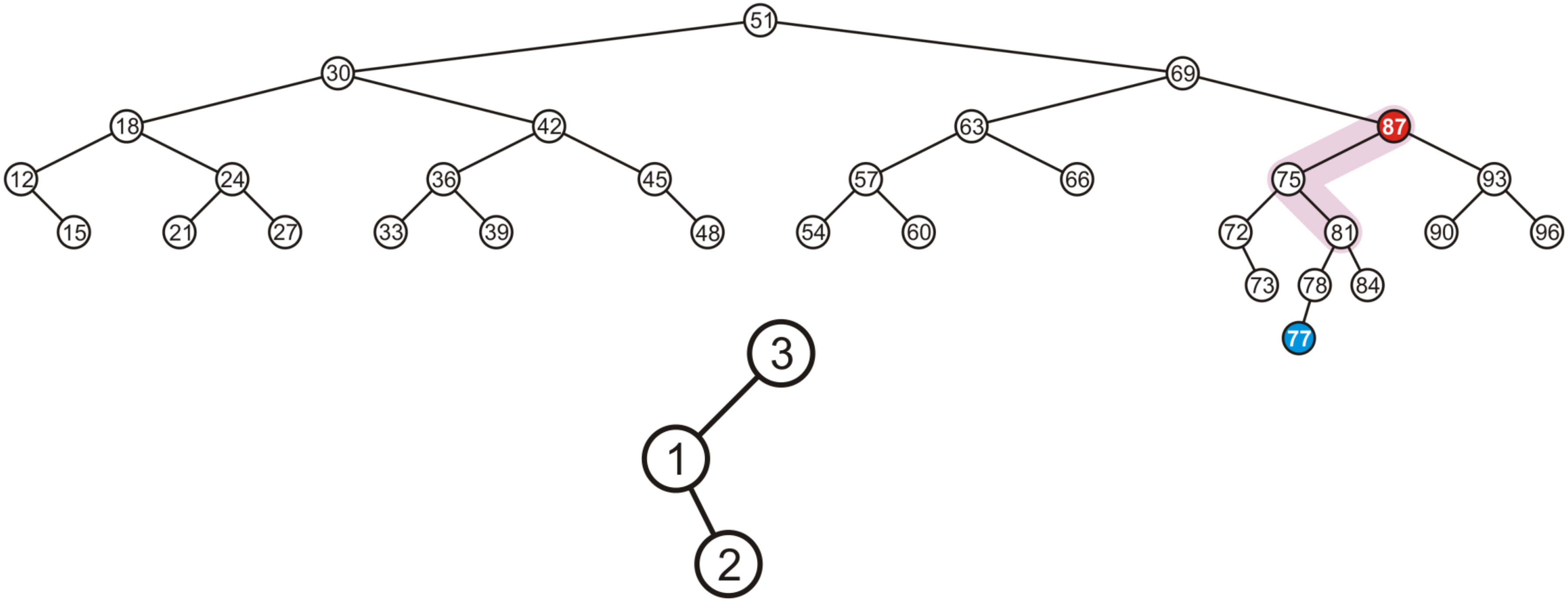
- A left-right imbalance



Insertion

The node 87 is unbalanced

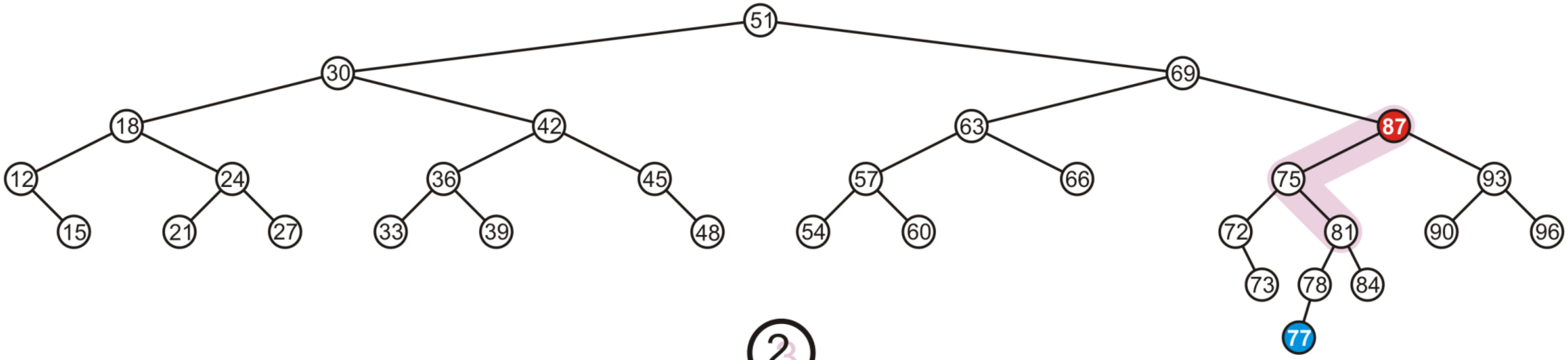
- A left-right imbalance

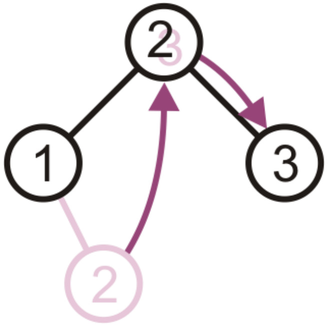


Insertion

The node 87 is unbalanced

- A left-right imbalance

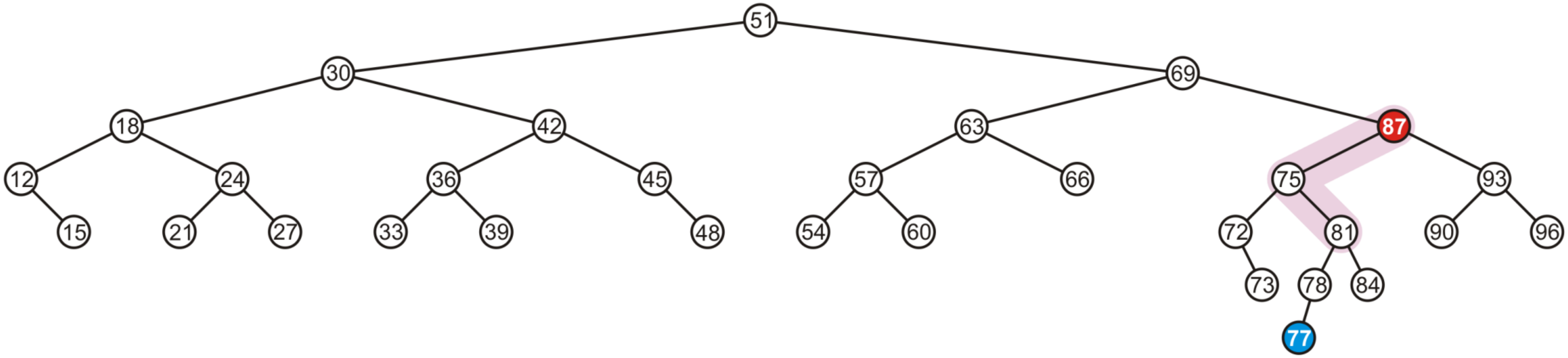




Insertion

The node 87 is unbalanced

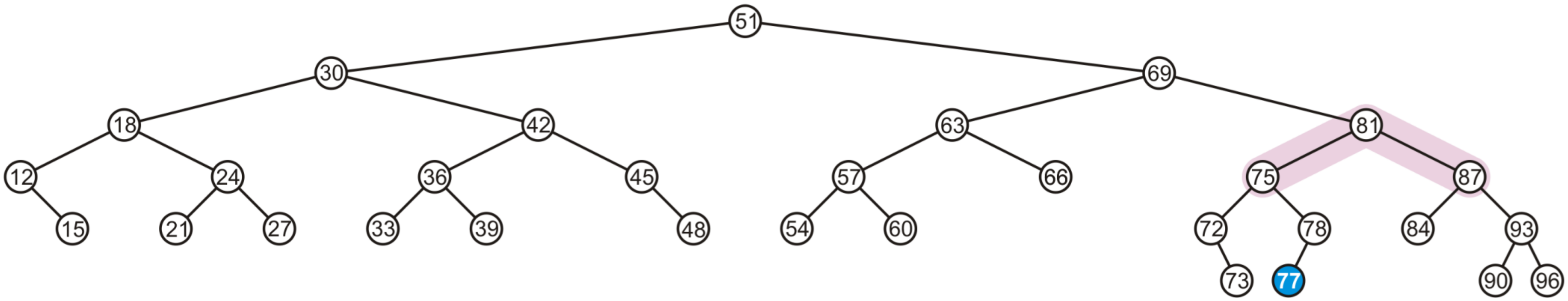
- A left-right imbalance
- Promote the intermediate node to the imbalanced node
- 81 is that value



Insertion

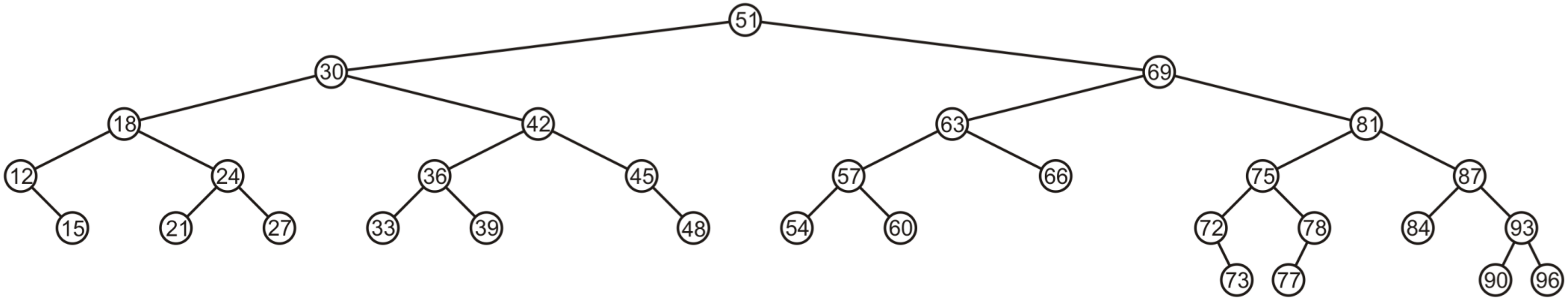
The node 87 is unbalanced

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



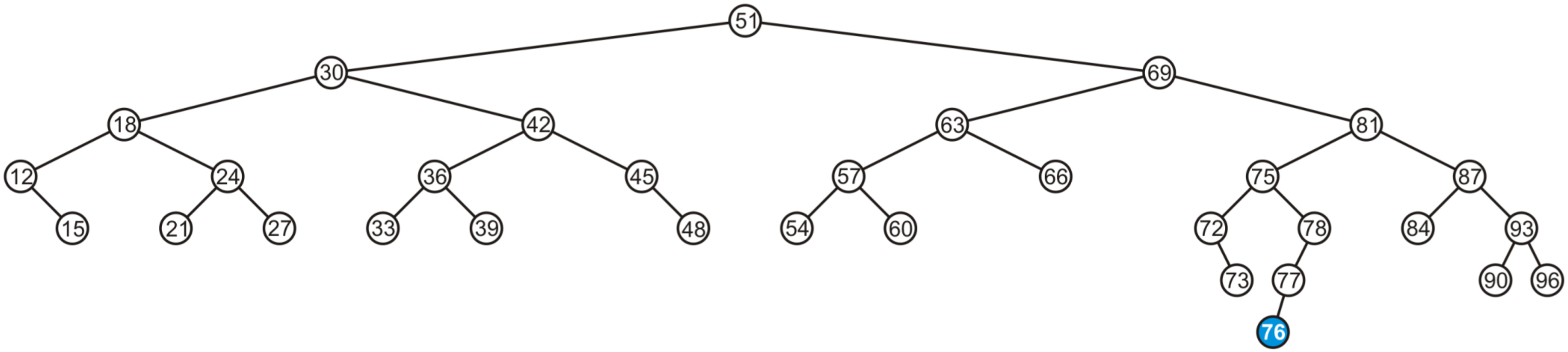
Insertion

The tree is balanced



Insertion

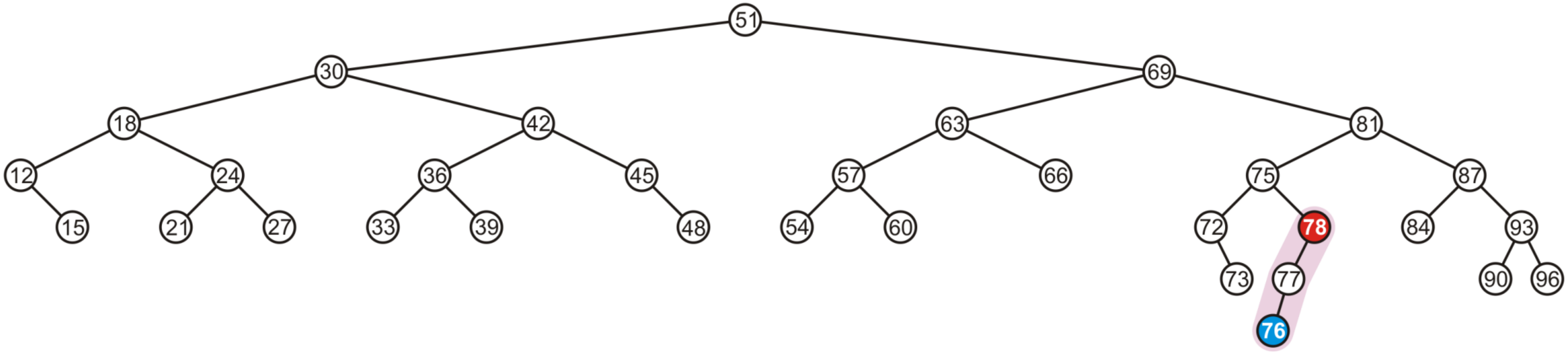
Insert 76



Insertion

The node 78 is unbalanced

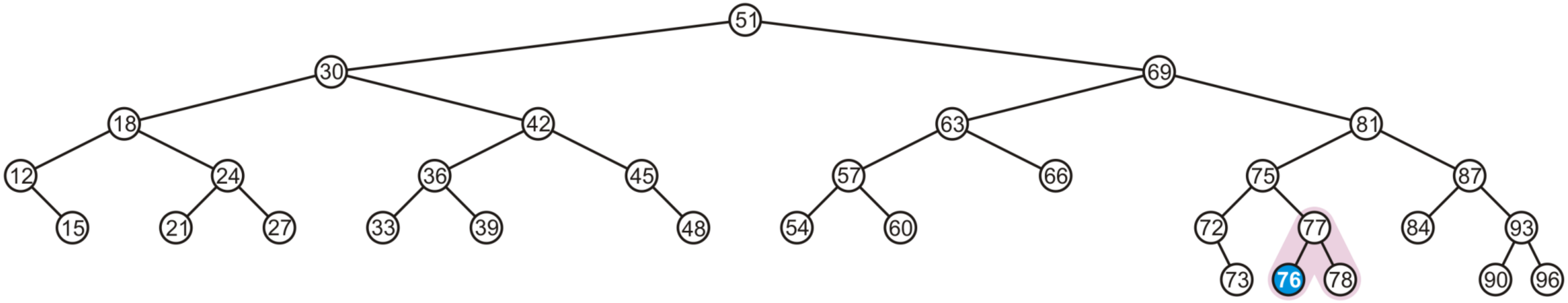
- A left-left imbalance



Insertion

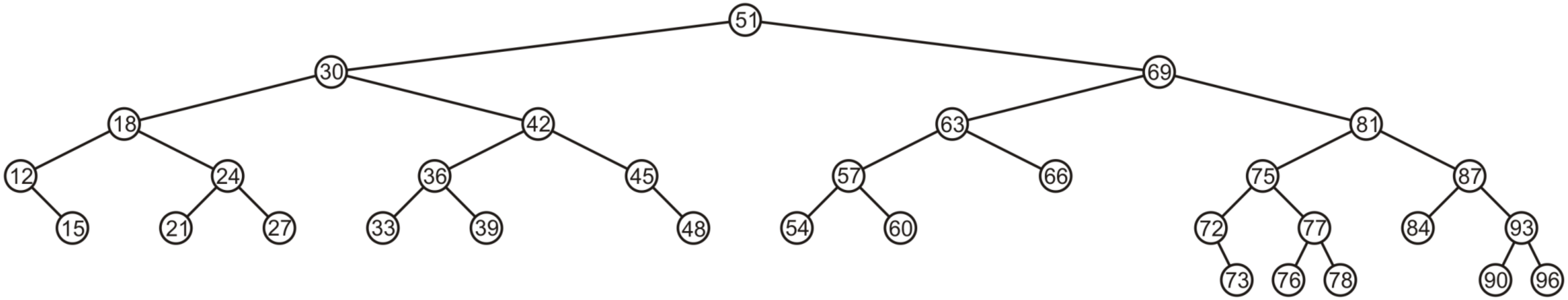
The node 78 is unbalanced

- Promote 77



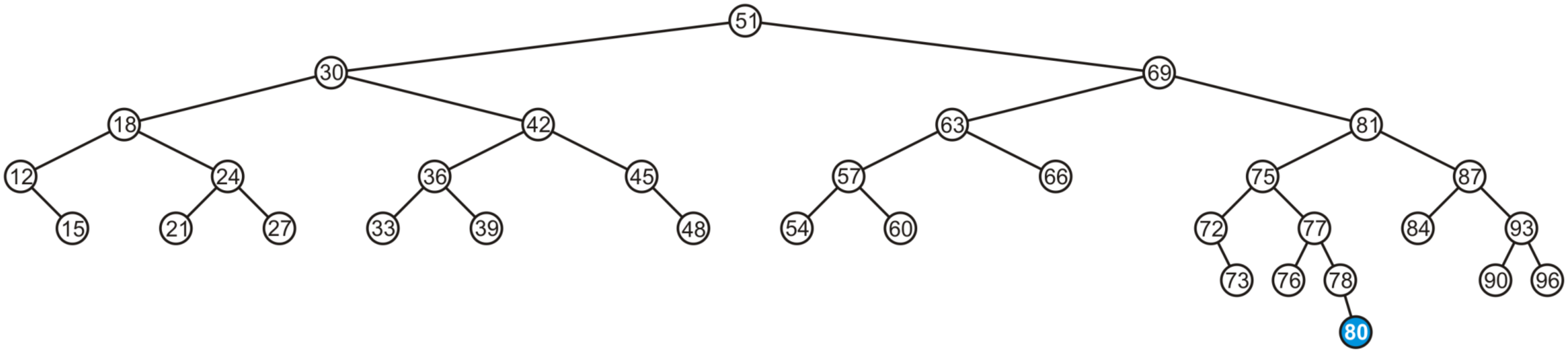
Insertion

Again, balanced



Insertion

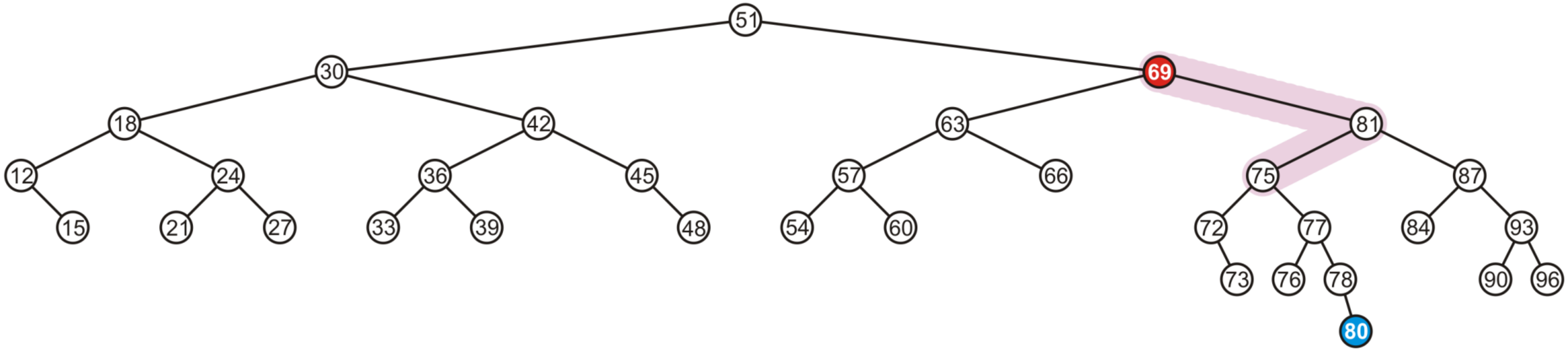
Insert 80



Insertion

The node 69 is unbalanced

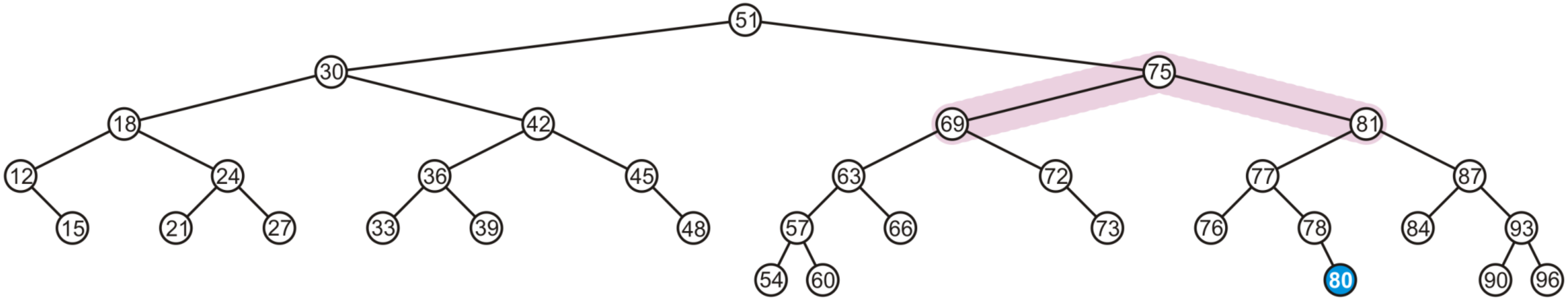
- A right-left imbalance
- Promote the intermediate node to the imbalanced node



Insertion

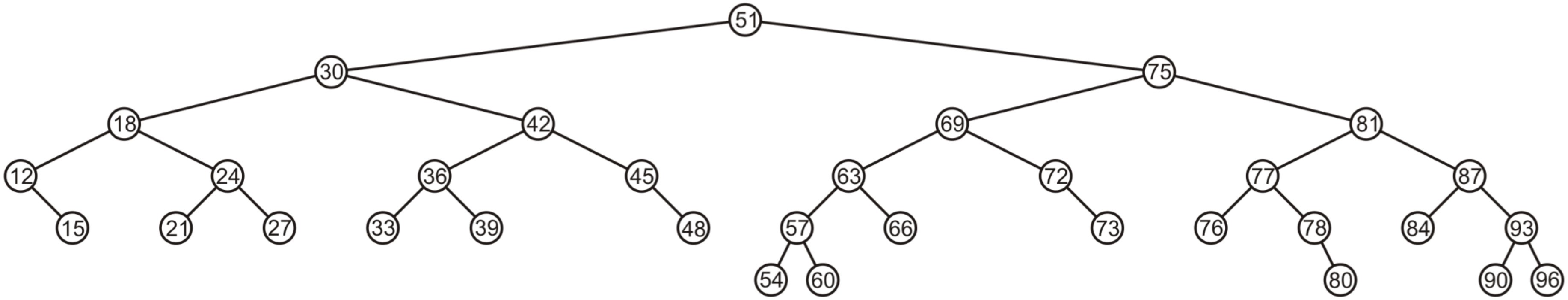
The node 69 is unbalanced

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



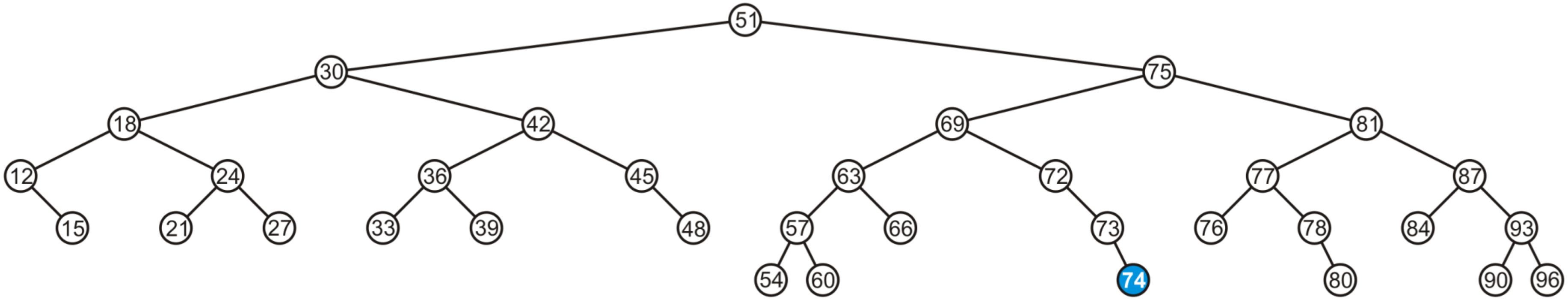
Insertion

Again, balanced



Insertion

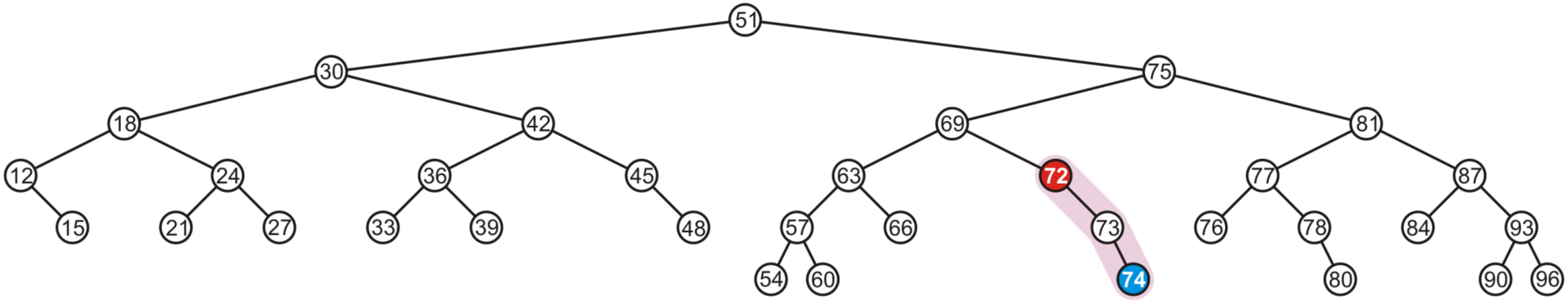
Insert 74



Insertion

The node 72 is unbalanced

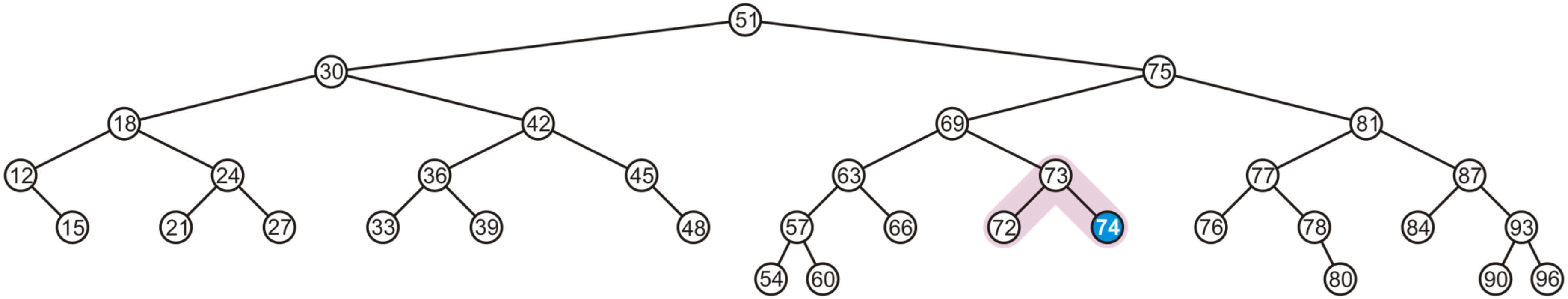
- A right-right imbalance



Insertion

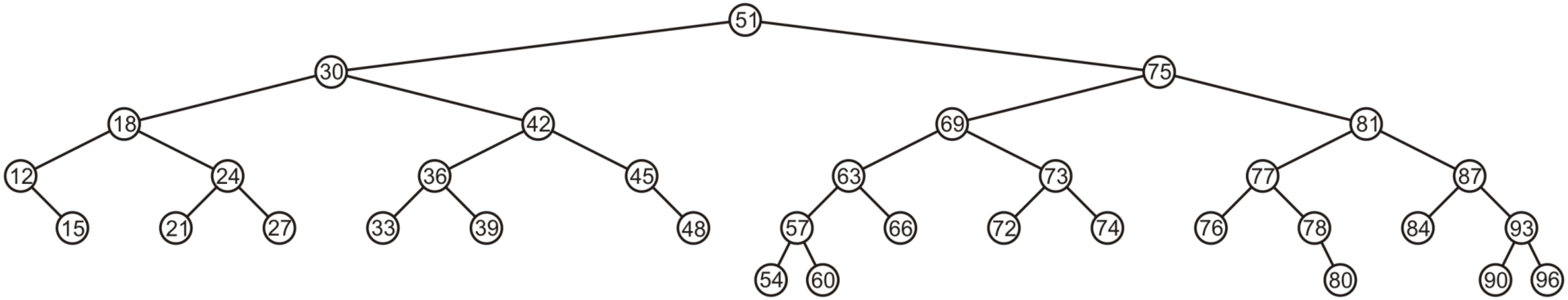
The node 72 is unbalanced

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



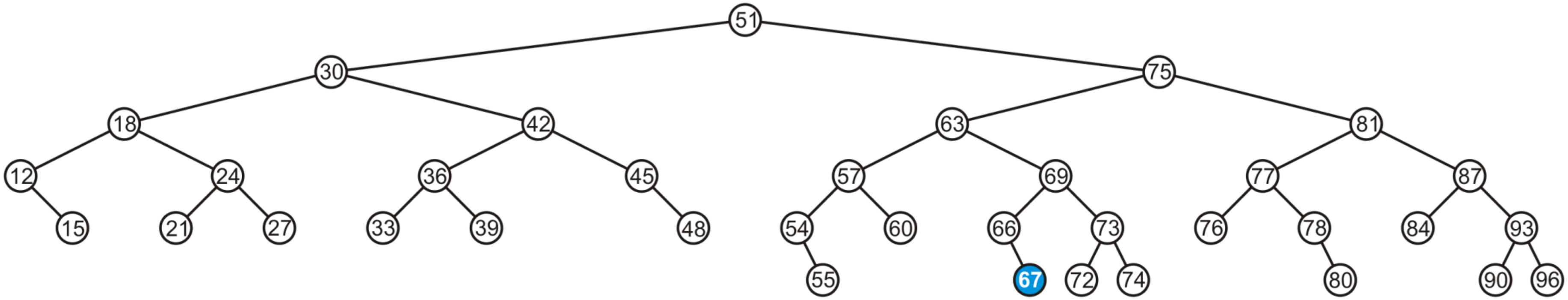
Insertion

Again, balanced



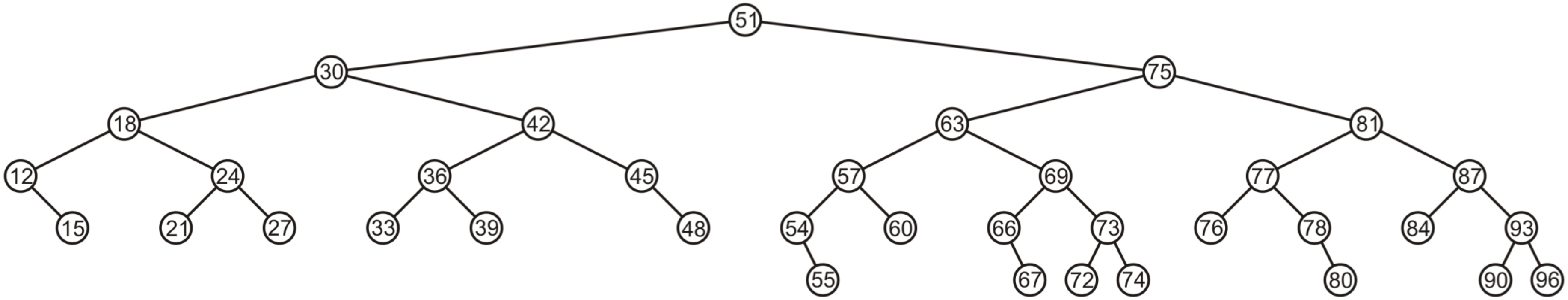
Insertion

Insert 67



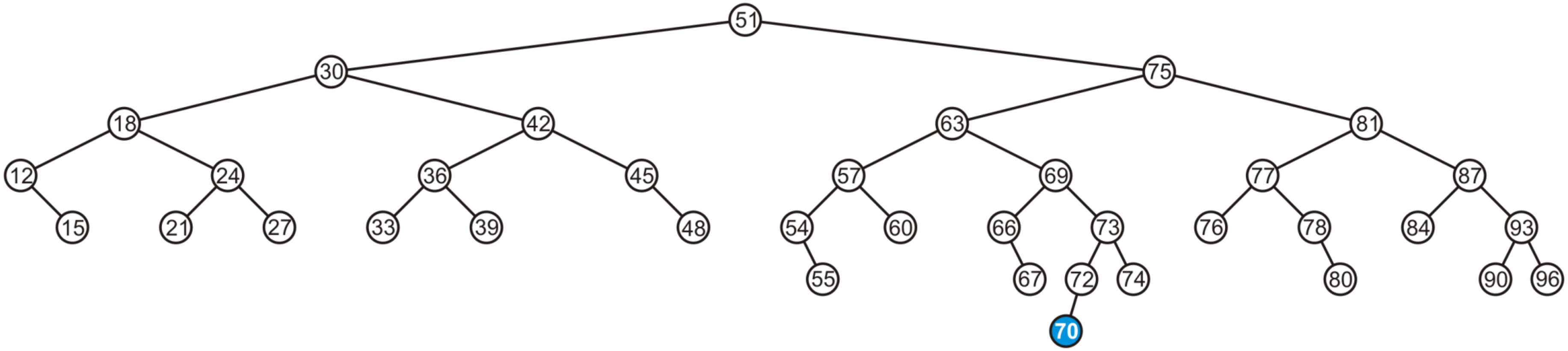
Insertion

Again, balanced



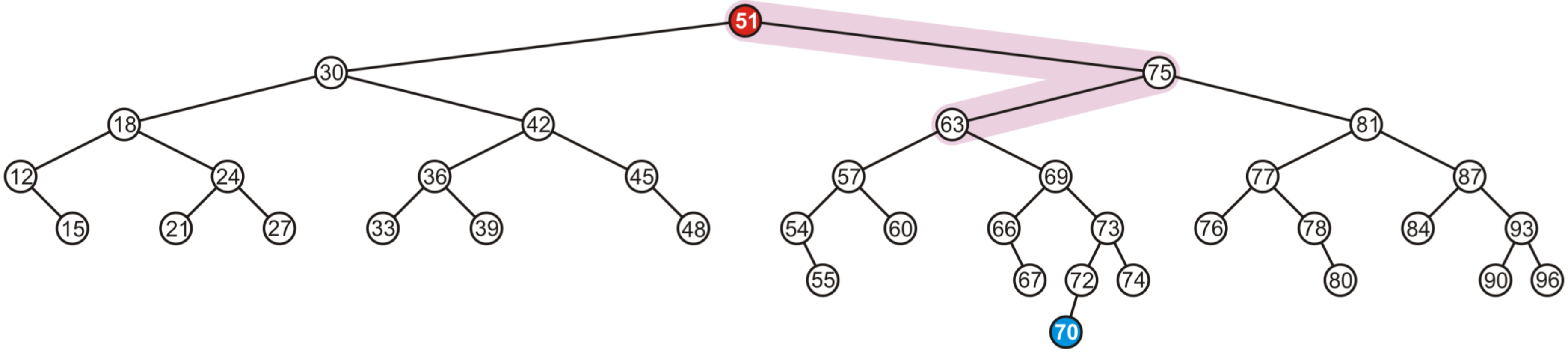
Insertion

Insert 70



Insertion

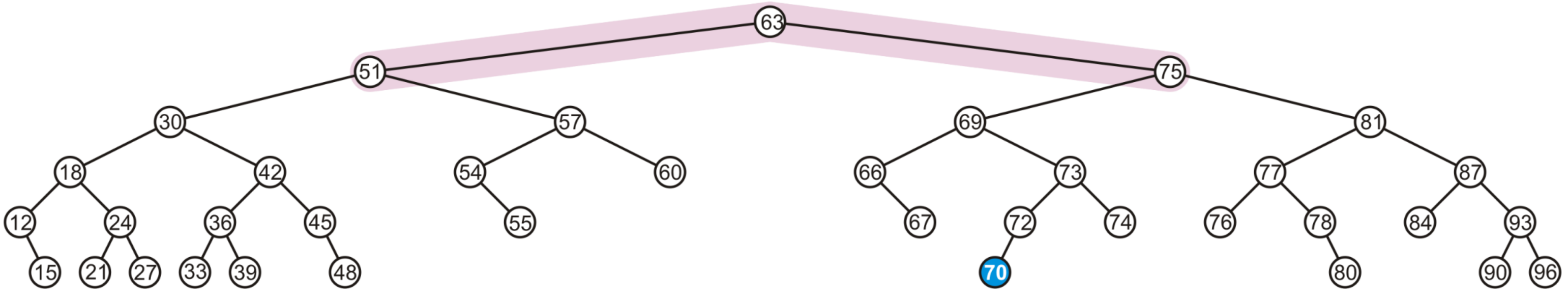
- The root node is now imbalanced
  - A right-left imbalance



Insertion

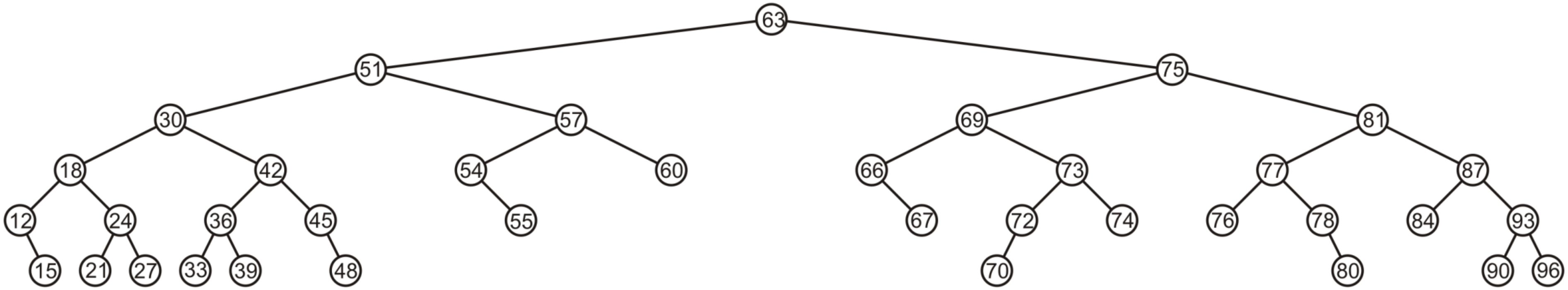
The root node is imbalanced

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



Insertion

The result is balanced



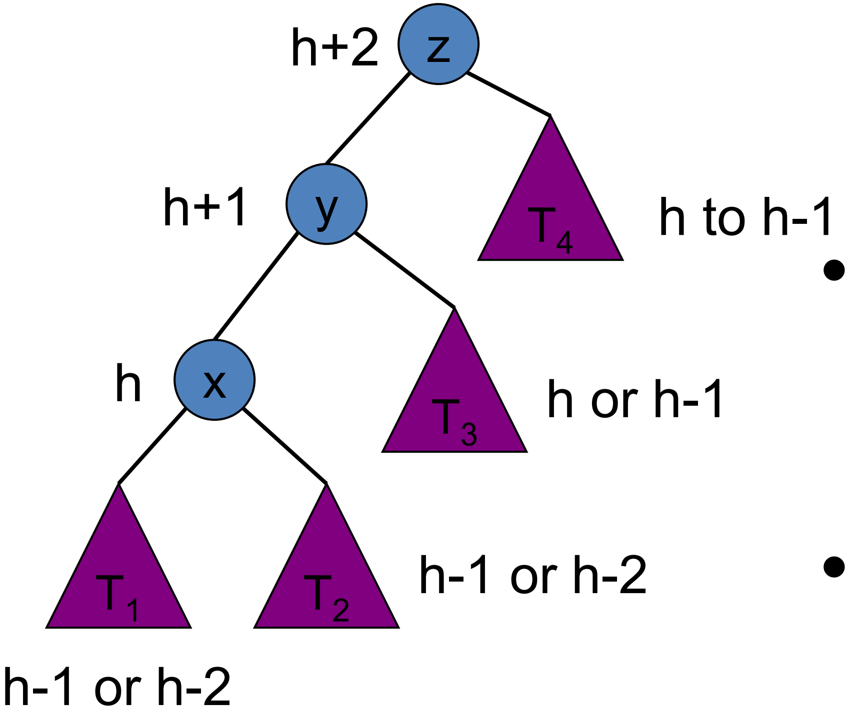
Deletion

- When deleting a node in a BST, we either delete a leaf or a node with only one child.
- In an AVL tree if a node has only one child then that child is a leaf.
- Hence in an AVL tree we either delete a leaf or the parent of a leaf.

Deletion(2)

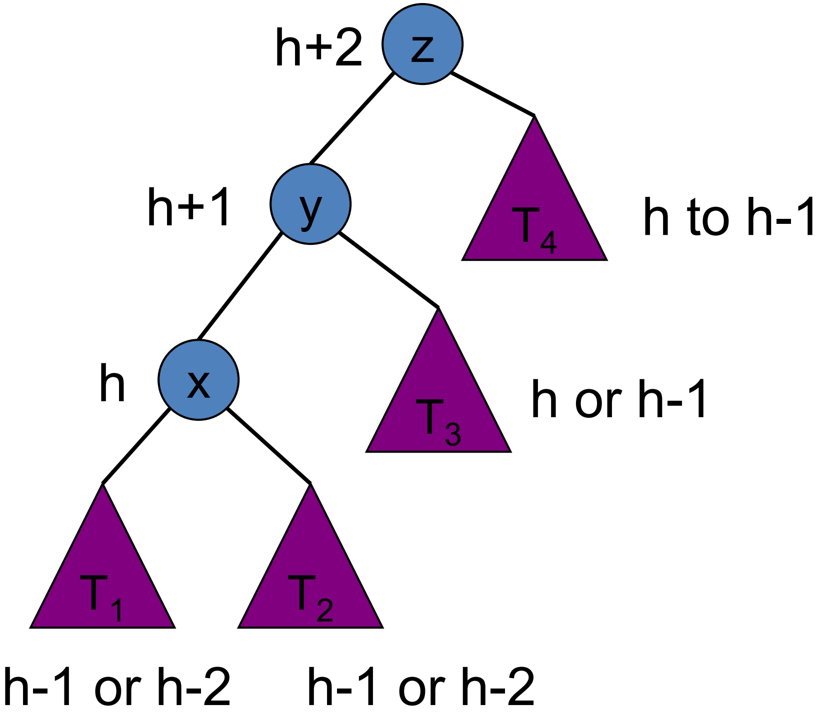
- Let w be the node deleted.
- Let *z* be the first unbalanced node encountered while travelling up the tree from w. Also, let y be the child of z with larger height, and let x be the child of y with larger height (what if tie happens?).
- We perform rotations to restore balance at the subtree rooted at z.
- As this restructuring may upset the balance of another node higher in the tree, we must continue checking for balance until the root of T is reached

Deletion(3)



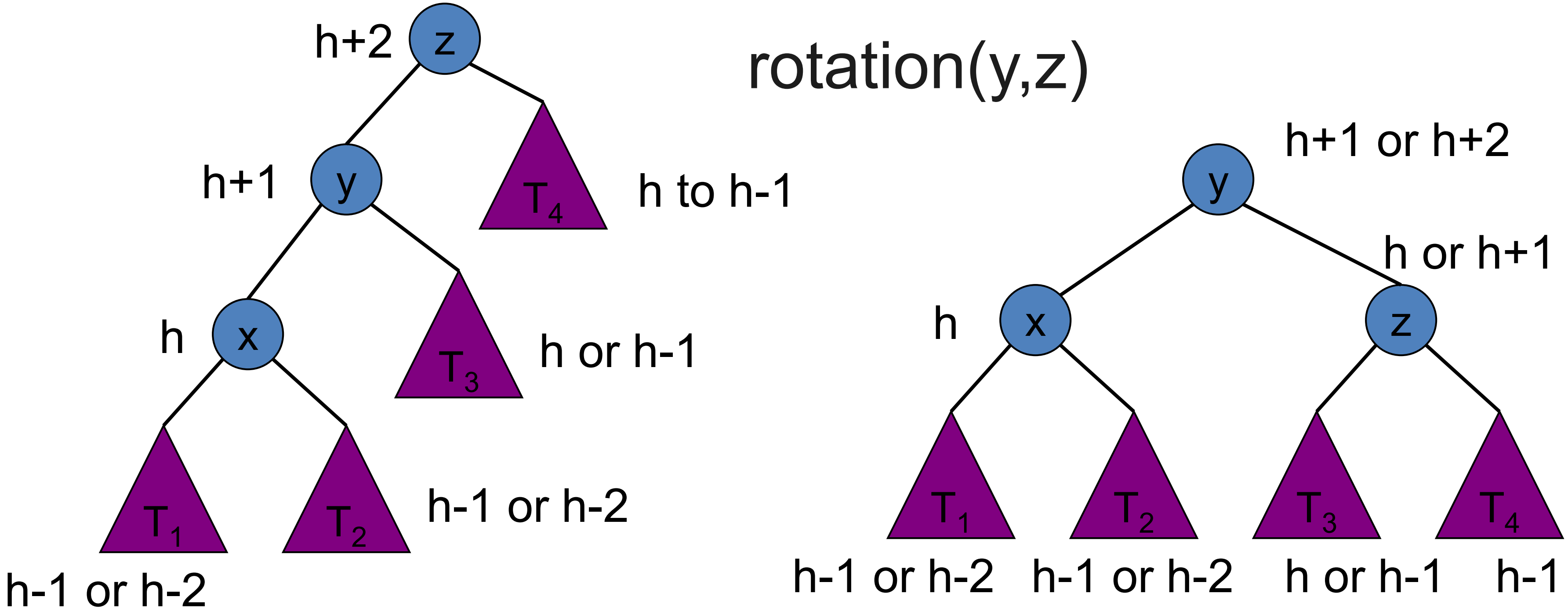
- Suppose deletion happens in subtree T_4 and its ht. reduces from h to h-1.
- Since z was balanced but is now unbalanced, ht(y) = h+1.
- x has larger ht. than T_3 and so ht(x)=h.
- Since y is balanced ht(T_3)= h or h-1

Deletion(4)



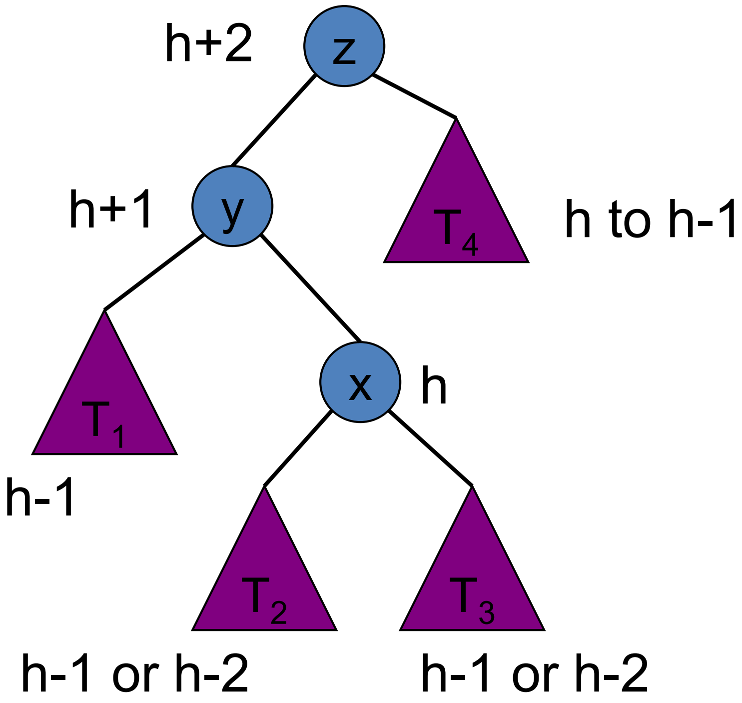
- Since ht(x)=h, and x is balanced ht(T_1), ht(T_2) is h-1 or h-2.
- However, both T_1 and T_2 cannot have ht. h-2

Single rotation (deletion)



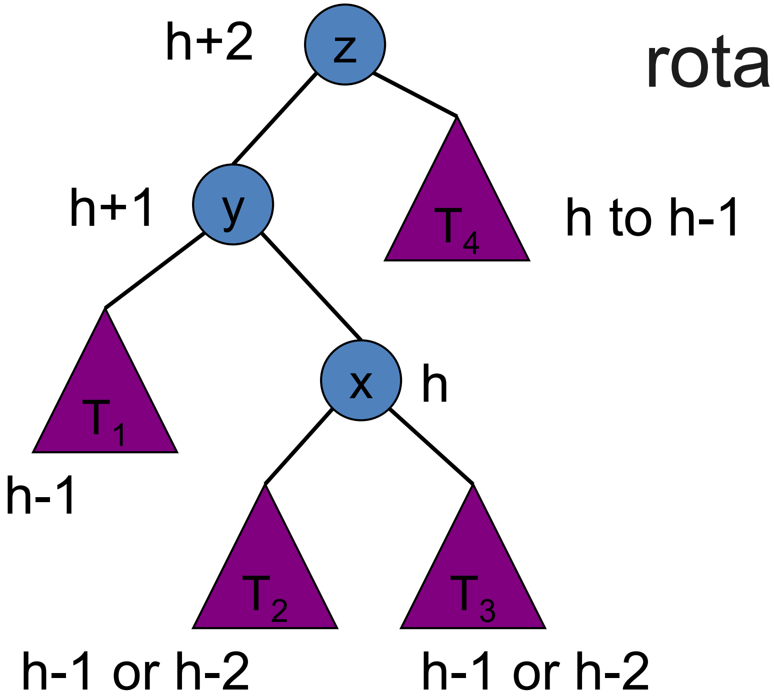
After rotation height of subtree might be 1 less than original height. In that case we continue up the tree

Deletion: another case

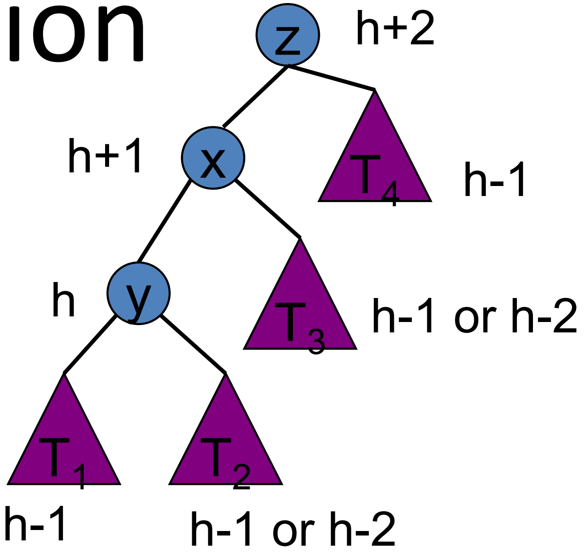


- As before we can claim that ht(y)=h+1 and ht(x)=h.
- Since y is balanced ht(T_1) is h or h-1.
- If ht(T_1) is h then we would have picked x as the root of T_1.
- So ht(T_1)=h-1

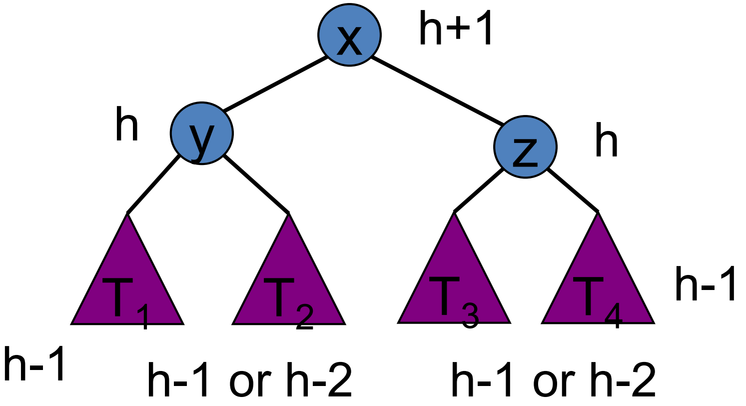
Double rotation



rotation(x,y)



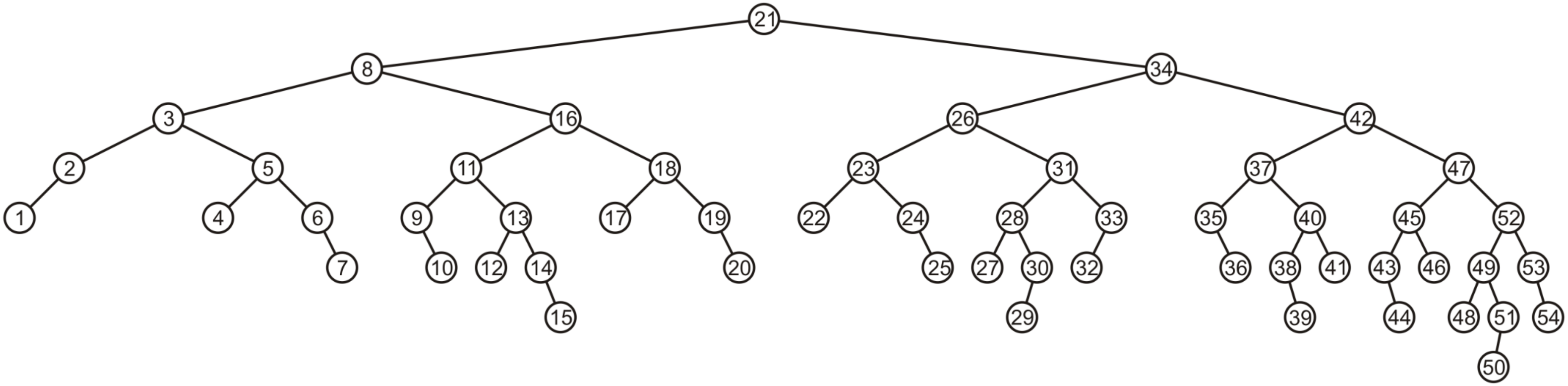
rotation(x,z)



Final tree has height less than original tree. Hence we need to continue up the tree

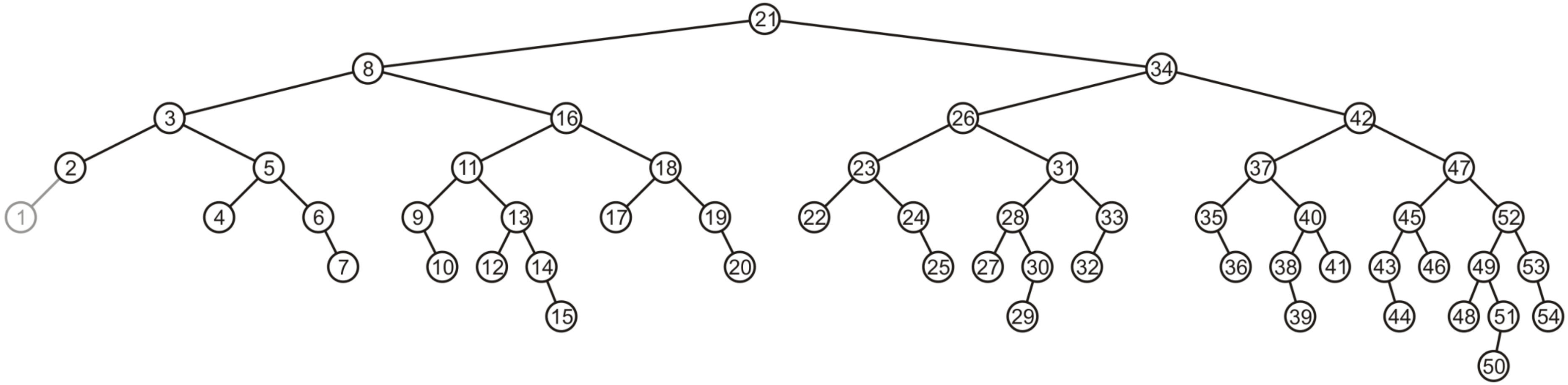
Deletion

Consider the following AVL tree



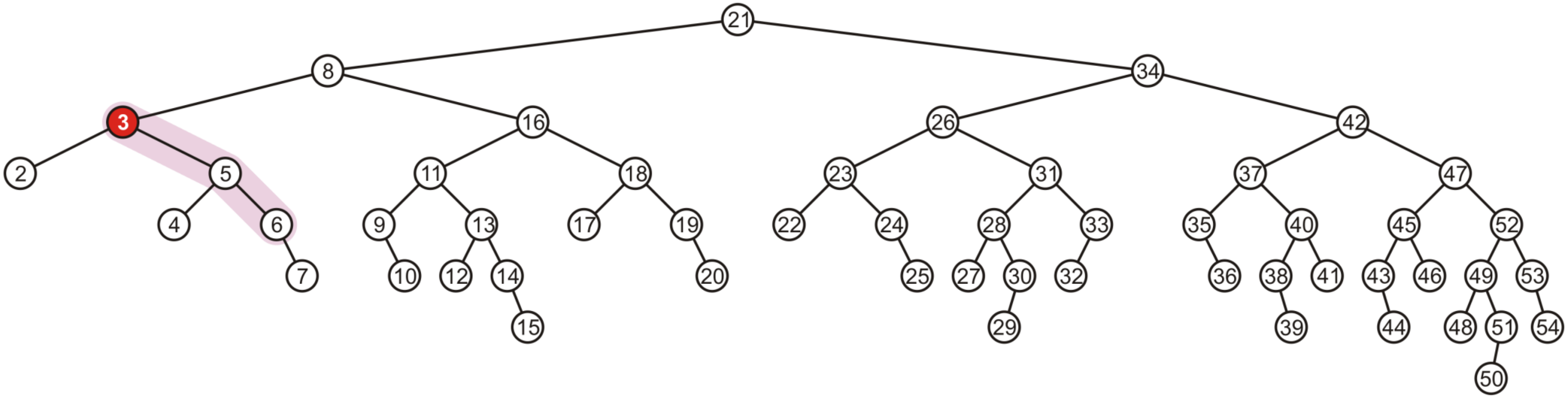
Deletion

Suppose we erase the front node: 1



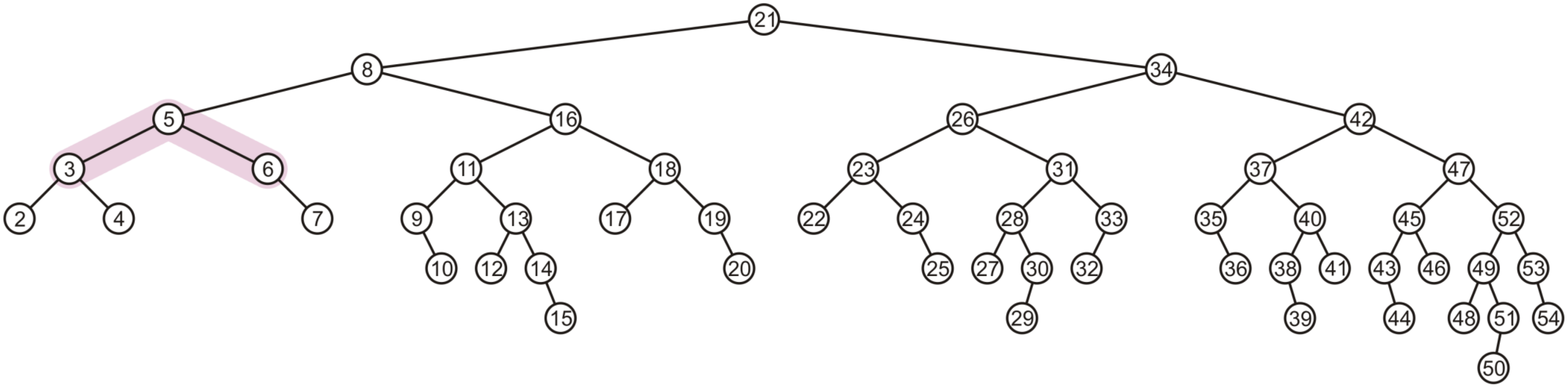
Deletion

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



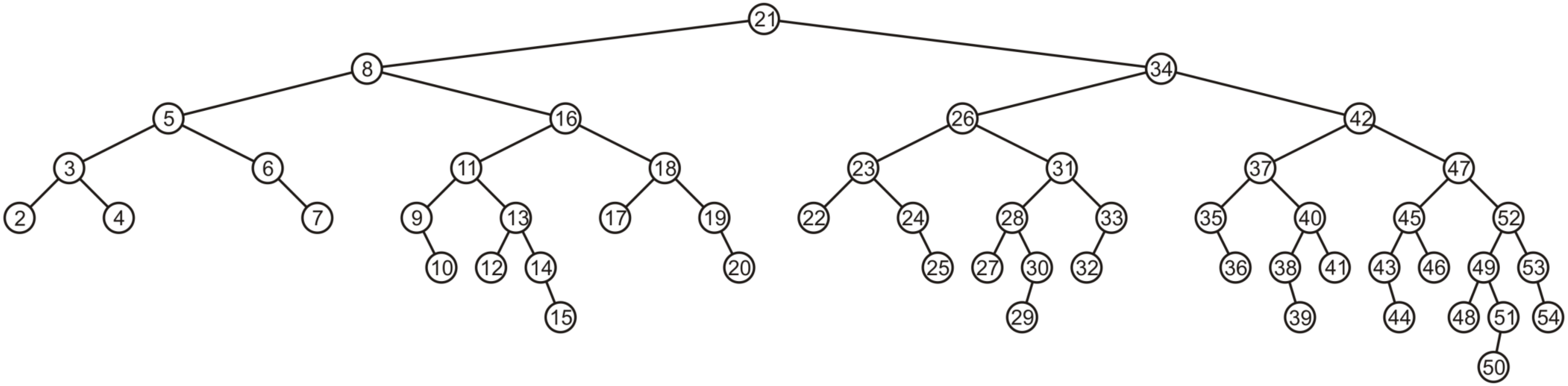
Deletion

We can correct this with a simple balance



Deletion

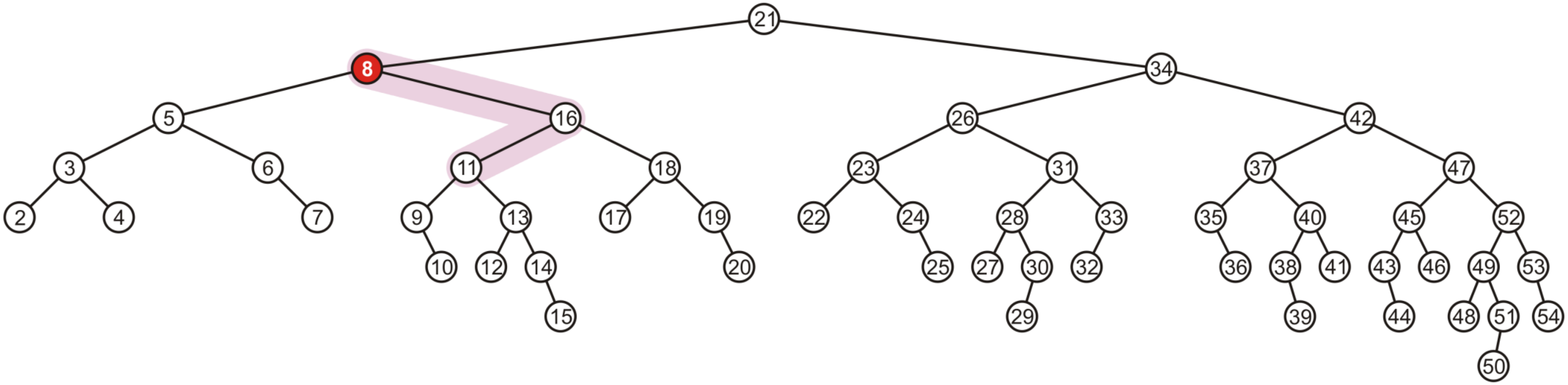
The node of that subtree, 5, is now balanced



Deletion

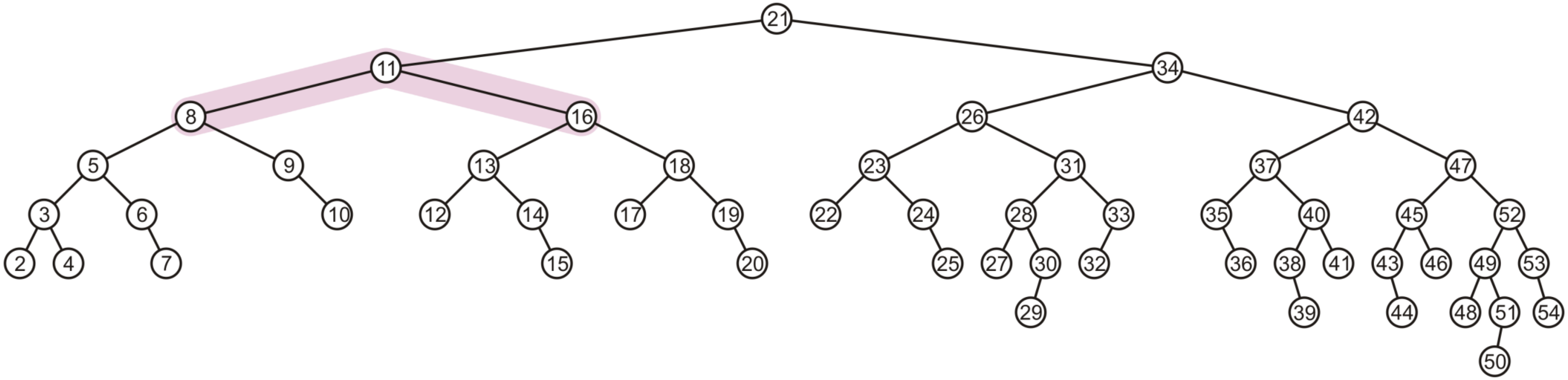
Recursing to the root, however, 8 is also unbalanced

- This is a right-left imbalance



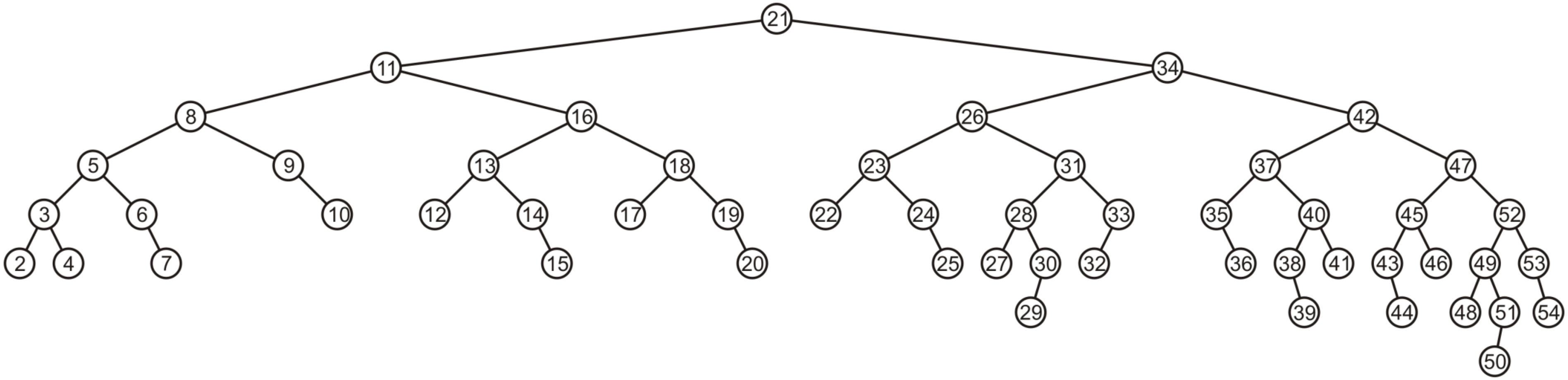
Deletion

Promoting 11 to the root corrects the imbalance



Deletion

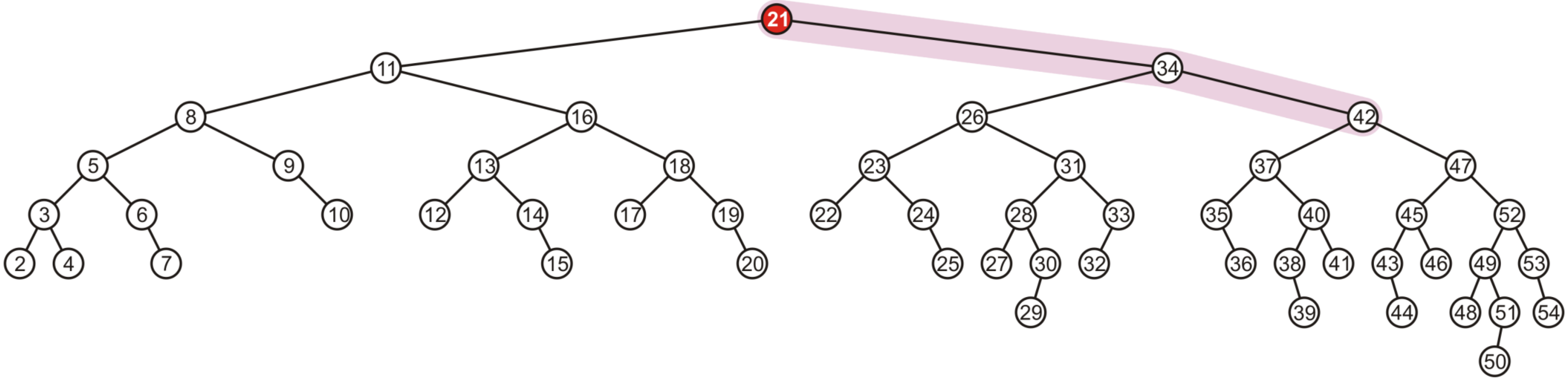
At this point, the node 11 is balanced



Deletion

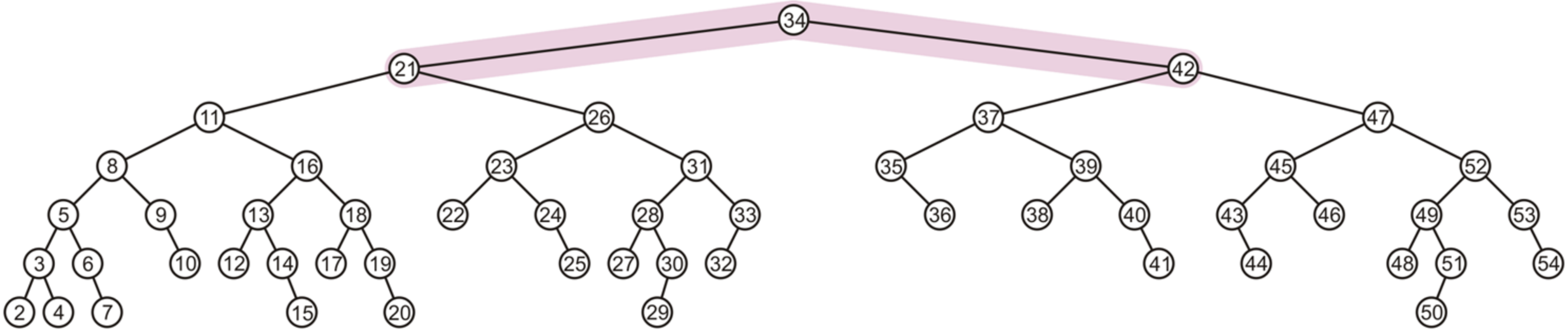
Still, the root node is unbalanced

- This is a right-right imbalance



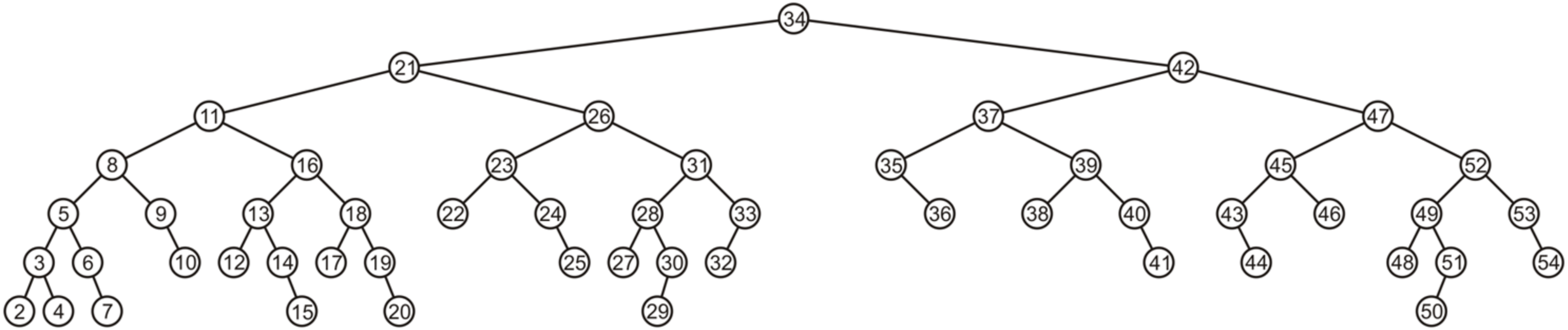
Deletion

Again, a simple balance fixes the imbalance



Deletion

The resulting tree is now AVL balanced



Running time of insertion & deletion

- Insertion
  - We perform rotation only once but might have to go O(log n) levels to find the unbalanced node.
  - So time for insertion is O(log n)
- Deletion
  - We need O(log n) time to delete a node.
  - Rebalancing also requires O(log n) time.
  - More than one rotation may have to be performed.

Pros and Cons of AVL Trees

Arguments for AVL trees:

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

Arguments against using AVL trees:

1. Difficult to program & debug; more space for balance factor.
2. Asymptotically faster but rebalancing costs time.
3. Most large searches are done in database systems on disk and use other structures (e.g. B-trees).