COP 3502: Computer Science I
Spring 2004

– Note Set 22 –
Self-Balancing Trees - AVL Trees

Instructor : Mark Llewellyn
markl@cs.ucf.edu
CC1 211, 823-2790
http://www.cs.ucf.edu/courses/cop3502/spr04

School of Electrical Engineering and Computer Science
University of Central Florida

Self-balancing Binary Search Trees

- As we have discussed within the context of the binary search tree, the performance of the algorithms which utilize the BST are dependent upon the height of the tree.
- We have looked at the DSW algorithm which takes an existing un-balanced BST and converts it into a perfectly balanced BST.
- In this set of notes we will look at a binary search tree which is self-balancing. In other words, each insertion and deletion operation will re-balance the tree. In this fashion, the tree is never un-balanced.
- There are many different types of self-balancing search trees. We'll examine just one of these in this course.

AVL Trees

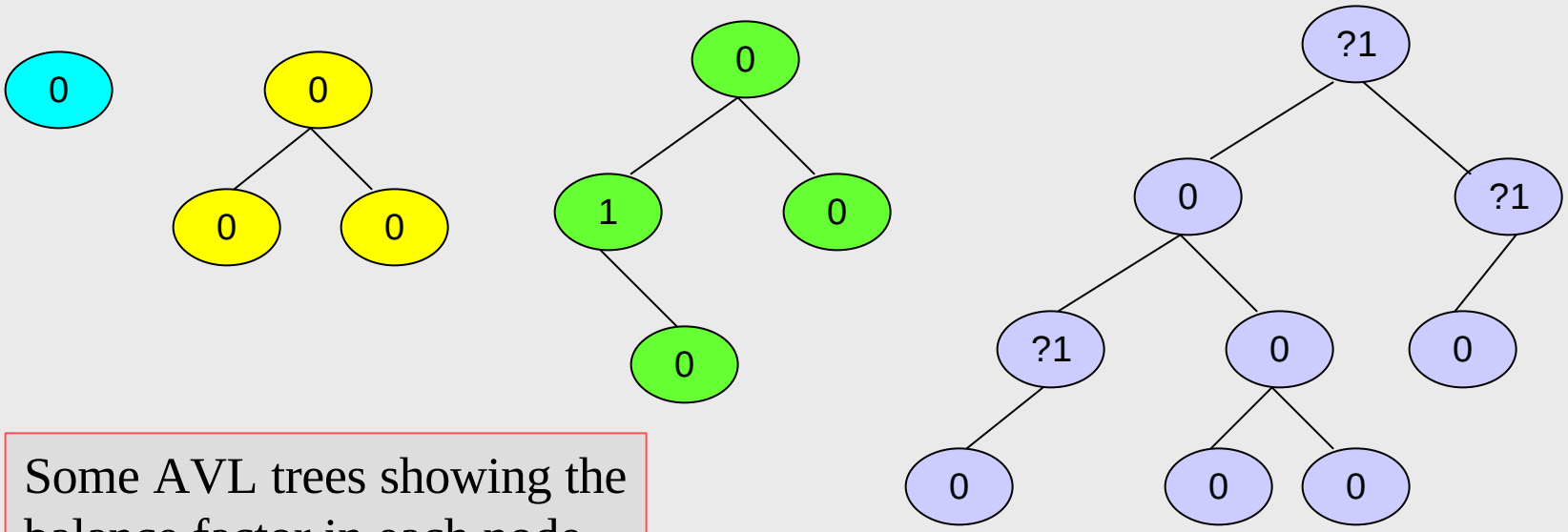
- AVL trees were originally called *admissible trees*, but were subsequently renamed after their discovery in 1962 by the Russian mathematicians Georgii M. Adel'son-Vel'skii and Evgenii M. Landis.
- An AVL tree is a binary search tree with a balance property (a structure property). The balance condition ensures that the height of the tree is O(log n) (its actually 1.44*log* n) where *n* is the number of nodes in the tree.
- All insertions, deletions, and searches in an AVL tree can be achieved in O(log n) time.
- In a completely balanced tree, the left and right subtrees of any node would have the same height. Since this can only occur for trees which are full, this definition is too restrictive for general search trees. The AVL tree relaxes this restriction to allow the heights of any two subtrees to differ by no more than 1.

AVL Trees (cont.)

DEFINITION:

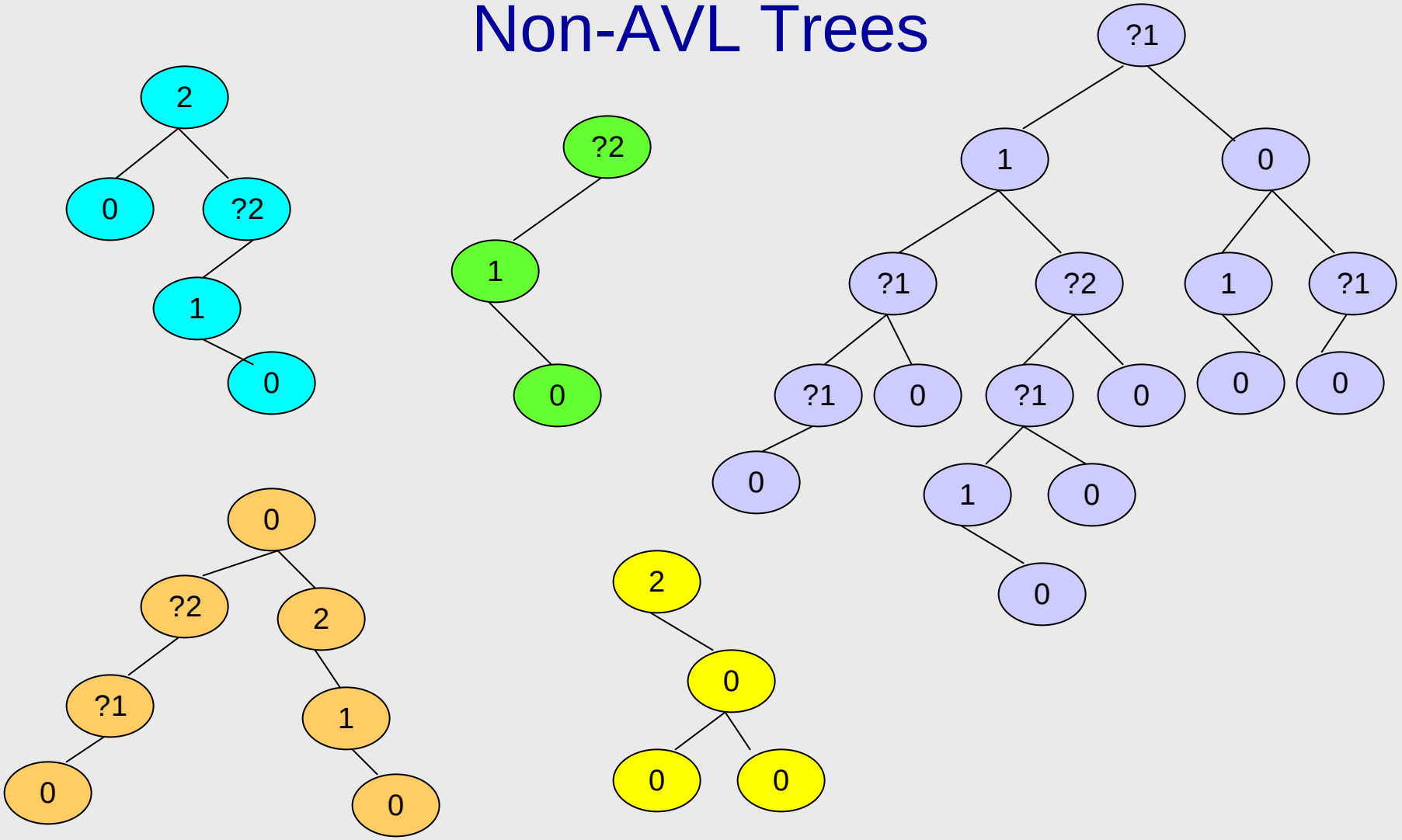
An AVL tree is a binary search tree in which the height of the left and right subtrees of the root differ by at most 1 and in which the left and right subtrees are also AVL trees.

With each node of an AVL tree is associated a *balance factor* that is determined by subtracting the height of the right subtree minus the height of the left subtree. For each node in an AVL tree the value of the balance factor is either –1, 0, or 1. The height of an empty subtree is assumed to be 0.



Some AVL trees showing the balance factor in each node.

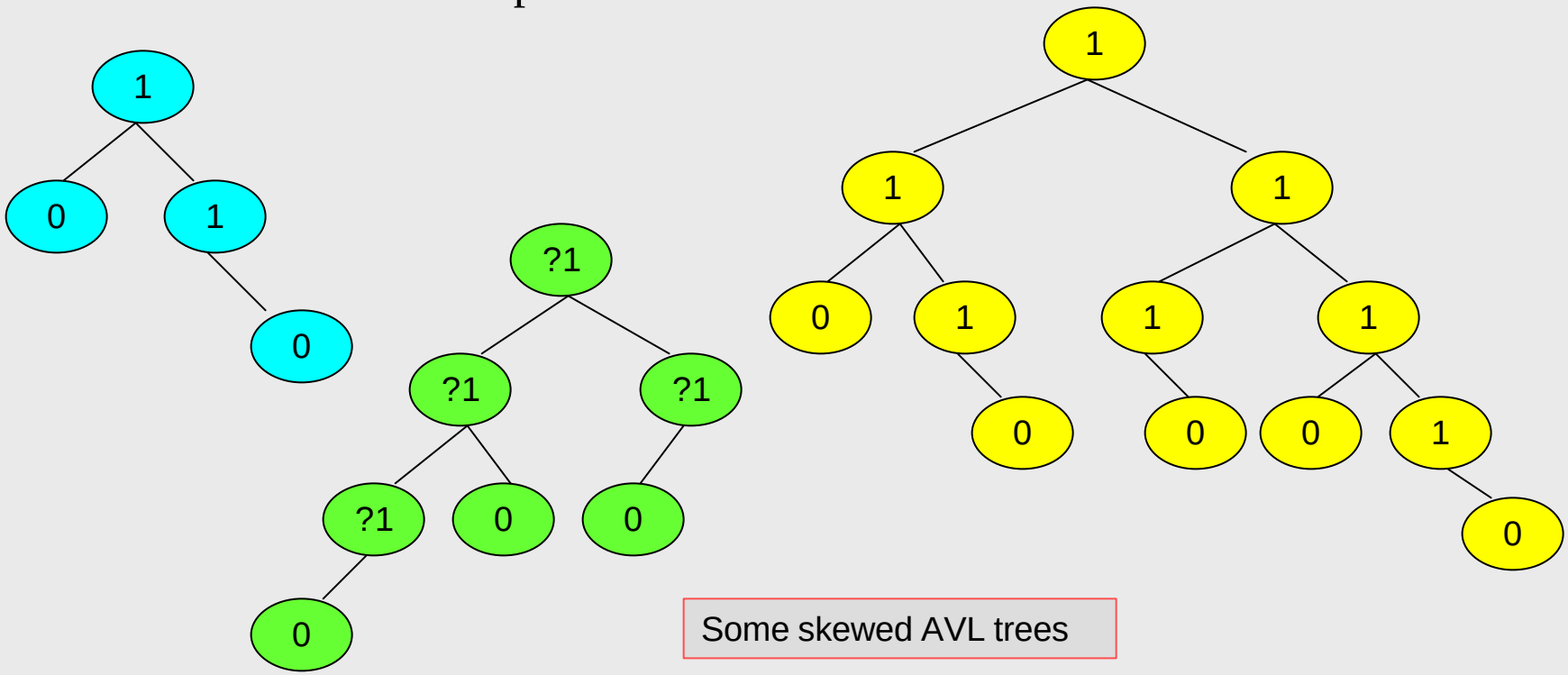
Non-AVL Trees



Some non-AVL trees showing the balance factor in each node.

Skewed AVL Trees

- Notice that the definition of an AVL tree does not require that all the leaf nodes be on the same or adjacent levels as was the case for a perfectly balanced binary search tree.
- It is possible to construct AVL trees which are quite skewed. Shown below are some examples.



Some skewed AVL trees

Insertion Into An AVL Tree

- The insertion of a new node into an AVL tree begins in the same fashion as insertion into a standard BST. Basically a search for the new value occurs which will be guaranteed to end on a null pointer in a leaf node with the actual insertion taking place in either the left or right subtree of that leaf node depending on its value compared to that in the leaf node. Once this is done, the balance must be checked and restored if the tree has become unbalanced.

- It often turns out that the new node can be inserted without changing the height of the subtree. If this occurs, then the balance of the root will not change.

- Even when the height of the subtree has increased, it may be that it was the shorter subtree that increased in height, so only the balance factor of the root will change.

- The only case that can cause difficulty occurs when the new node is added to a subtree of the root which is taller than the other subtree and the height of the taller subtree increases. This would cause the one subtree to have a height 2 more than the other, violating the AVL structure property.

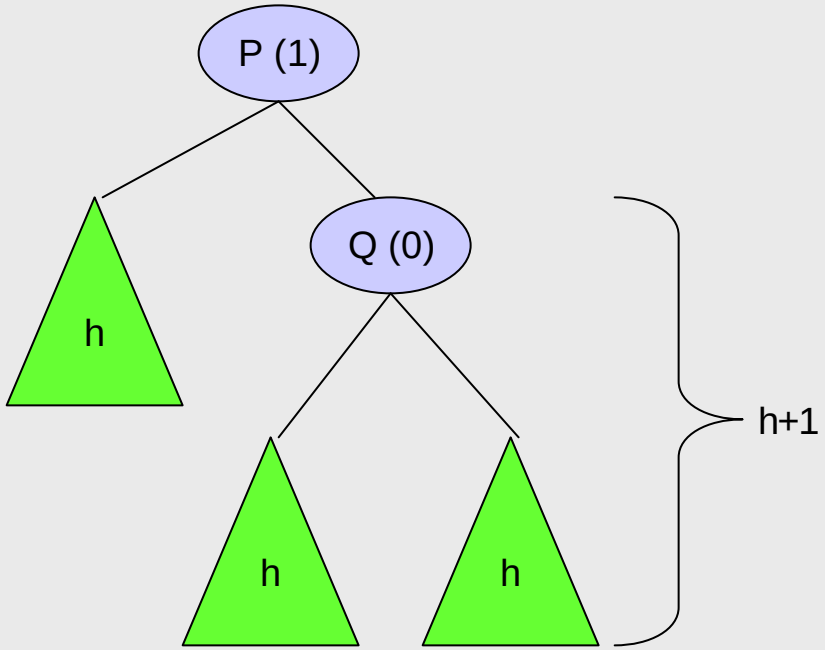
Insertion Into An AVL Tree (cont.)

- Thus, an AVL tree can become unbalanced due to an insertion in one of four ways (two of which are symmetric to the others).

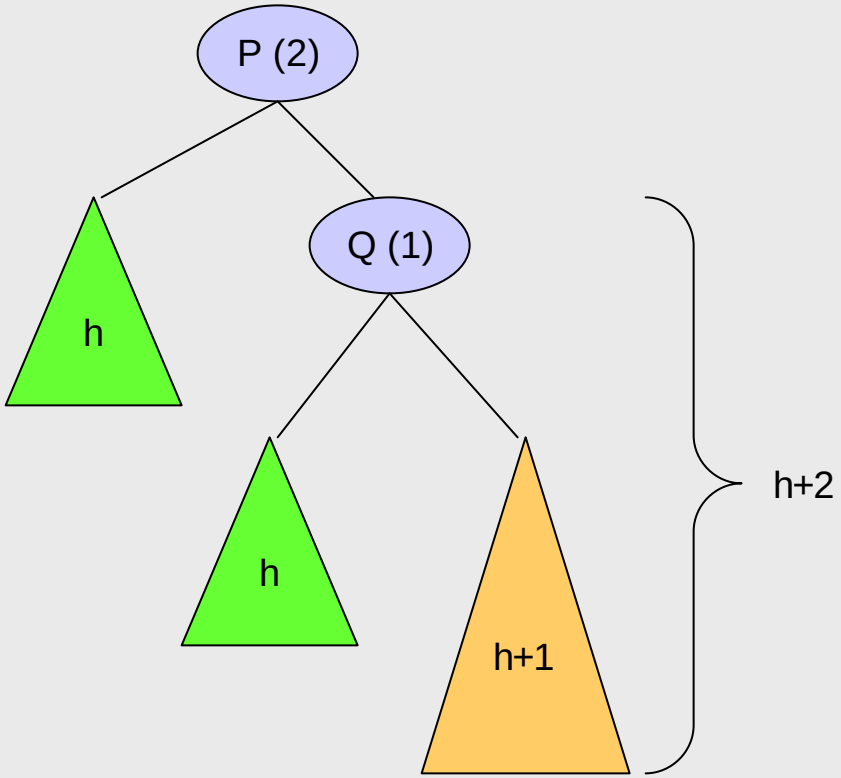
1. Inserting a new node into the right subtree of a right child.
    a. Symmetric case is insertion of new node into the left subtree of a left child.
2. Inserting a new node into the left subtree of a right child.
    a. Symmetric case is insertion of a new node into the right subtree of a left child.

- The first case is the simpler of the two to handle, so we'll examine this case first.

Case 1: Insertion Into the Right Subtree of a Right Child



Initial AVL tree before insertion

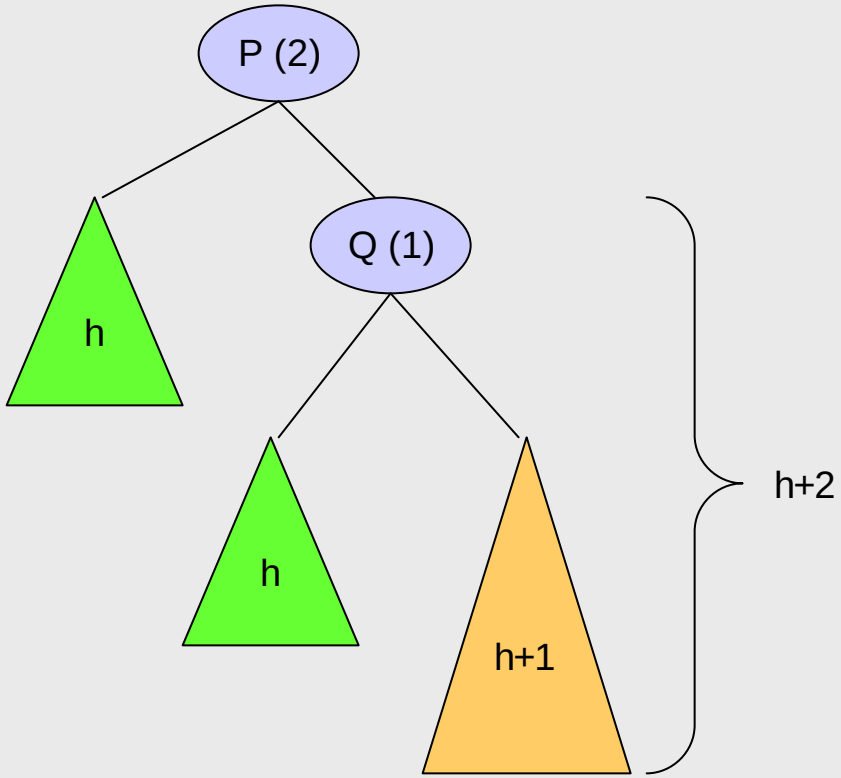


AVL tree after insertion into
right subtree of right child Q

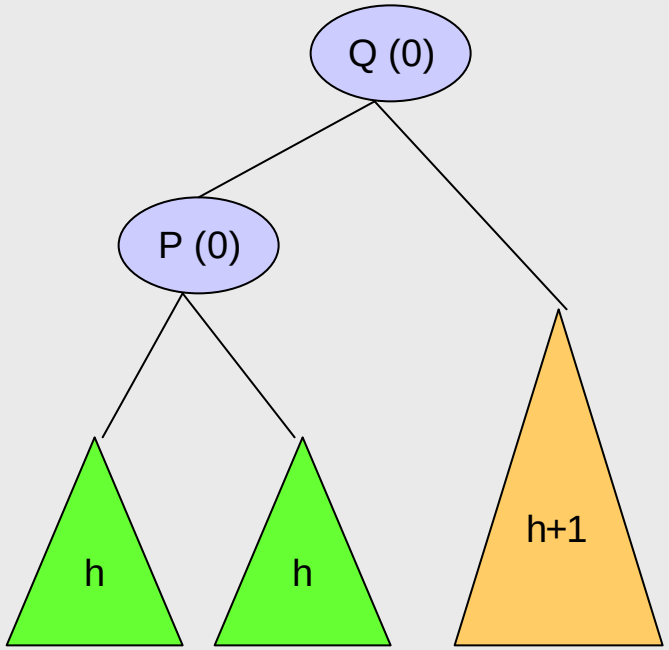
Handling the Imbalance for Case 1

- As the diagram on the previous slide clearly indicates, the AVL tree has become imbalanced due to the insertion of a new node into the right subtree of the right child of P.
- Note that the point of imbalance is at node P, the parent of the right child into whose subtree the insertion occurred.
- The question now becomes, how do we rebalance the tree after this insertion?
- The answer is via a rotation of Q about P. Notice that this will be a left rotation, effectively bring the subtree rooted at Q, up and to the left in the tree and pushing P down and to the left. This is illustrated in the next slide.

Single Left Rotation Handles Case 1 Imbalance

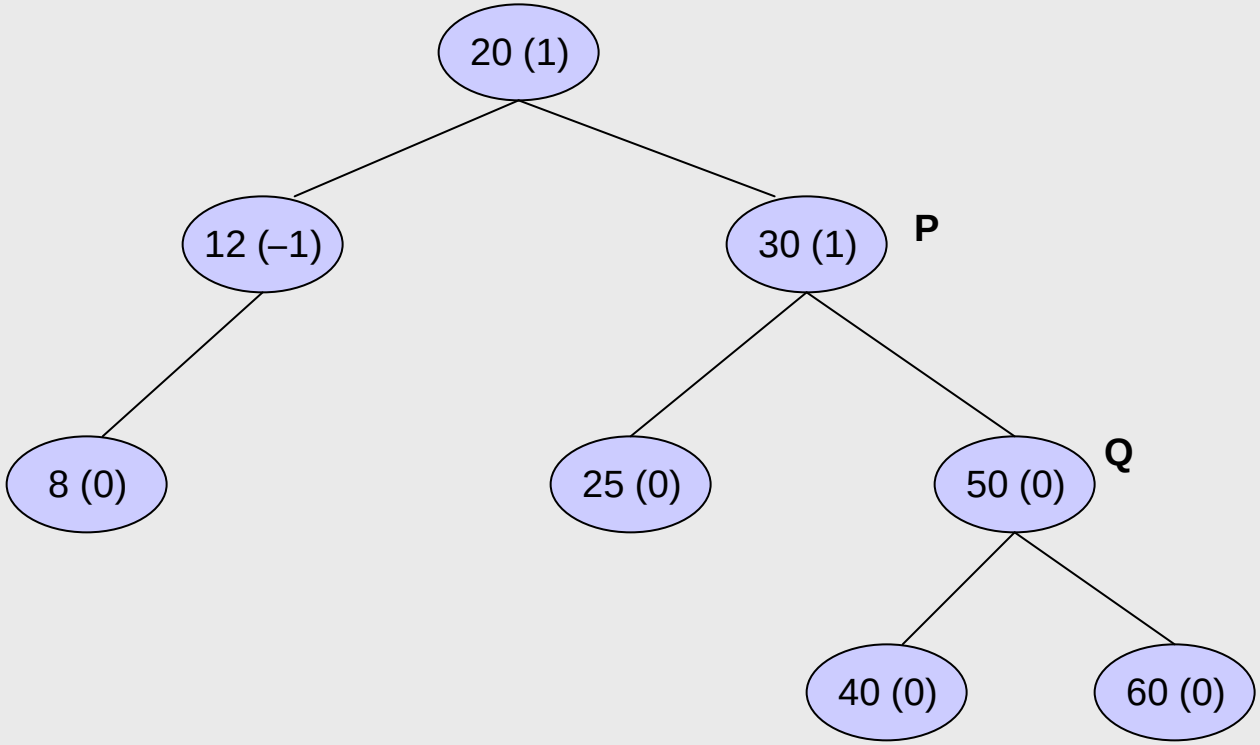


Result of a case 1 imbalance in an AVL tree after insertion into right subtree of right child Q



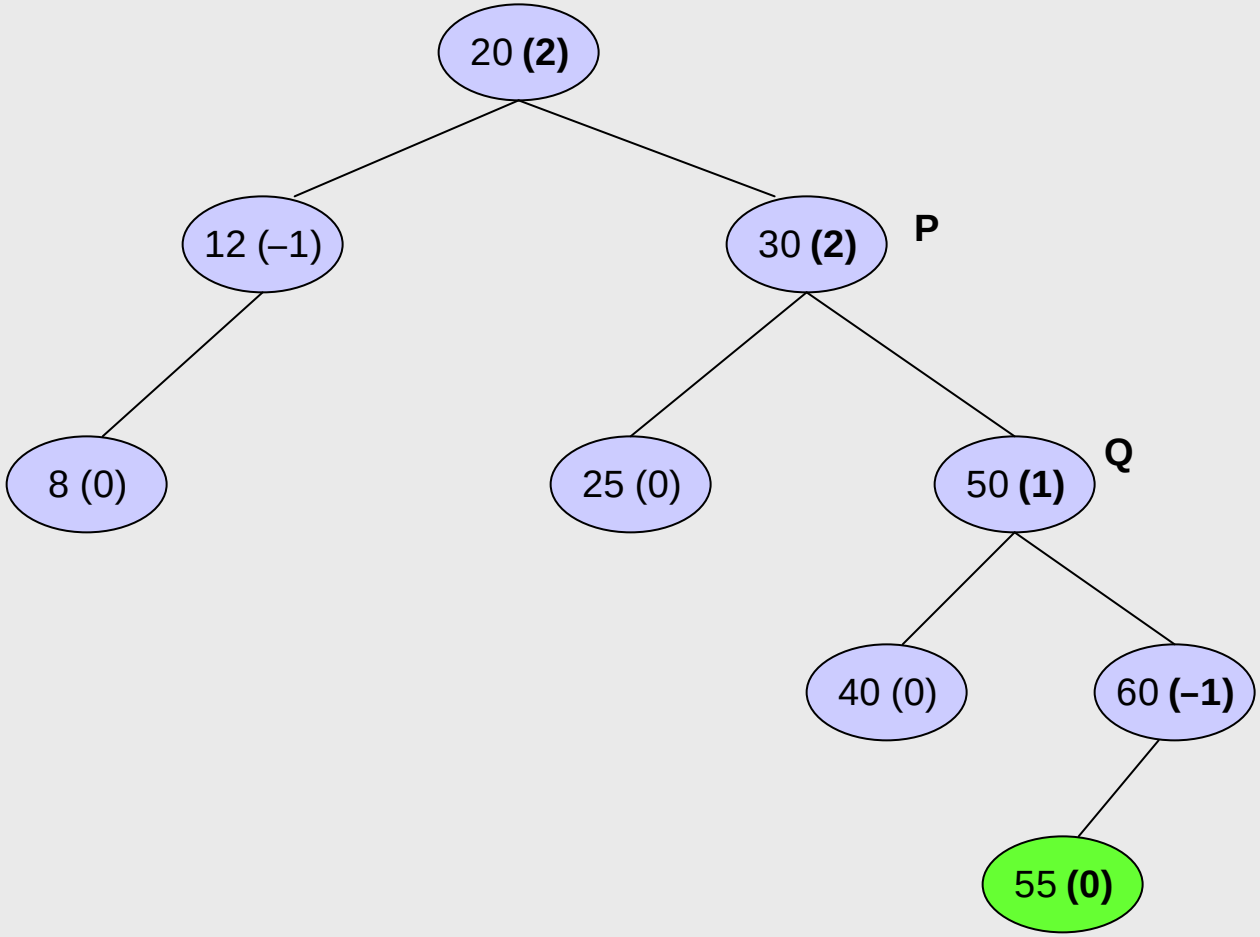
AVL tree after single left rotation of Q about P. Balance has been restored in the tree.

Example – Case 1 Insertion Imbalance



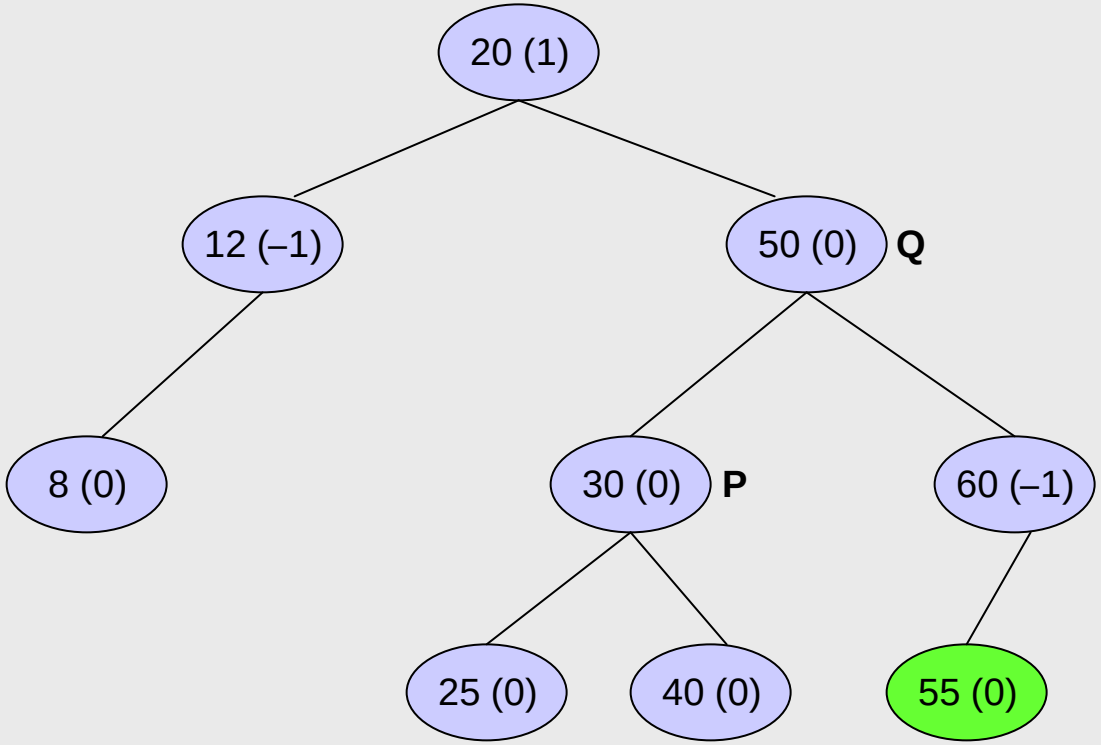
Initial AVL tree

Example – Case 1 Insertion Imbalance (cont.)



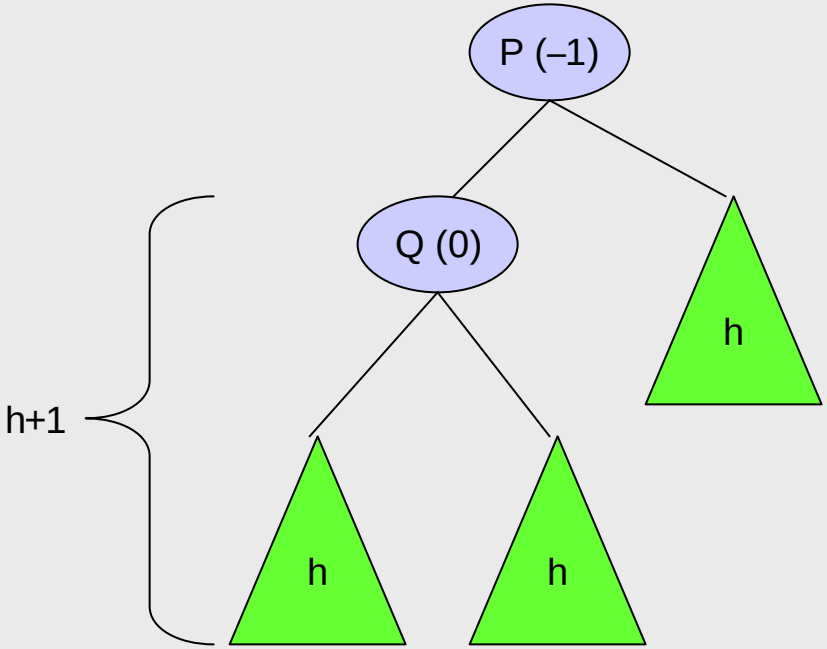
AVL tree after insertion of new node containing 55. Note the change in balance factors from the initial tree. Balance factors that have changed are shown in bold. Notice that they have changed all the way to the root of the tree.

Example – Case 1 Insertion Imbalance (cont.)

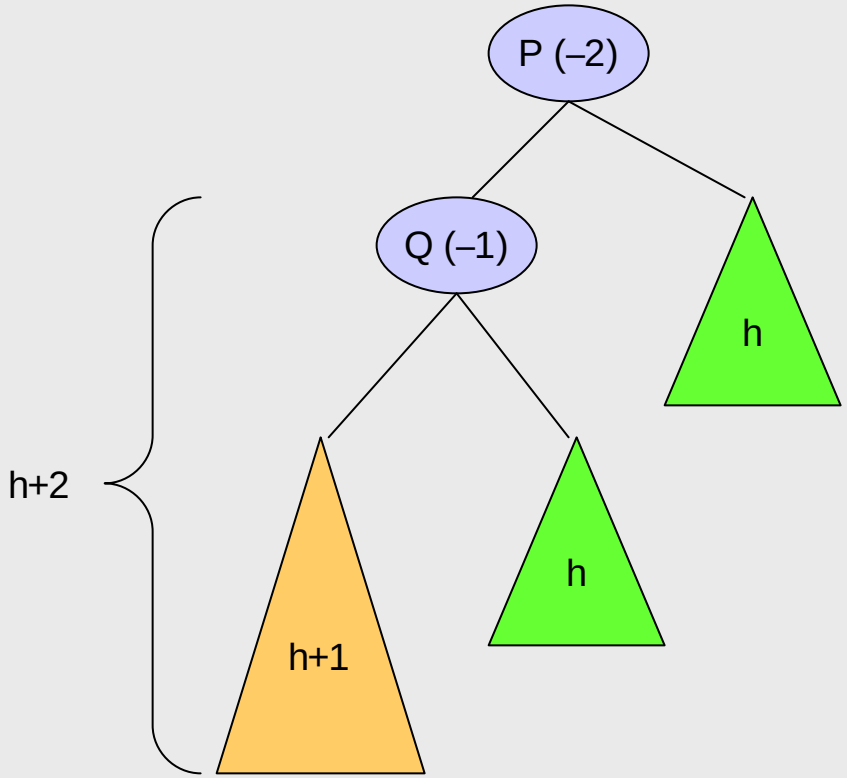


AVL tree after left rotation of 50 about 30 (Q about P). Note that the root of the tree is now balanced.

Symmetric Case 1a: Insertion Into the Left Subtree of a Left Child



Initial AVL tree before insertion

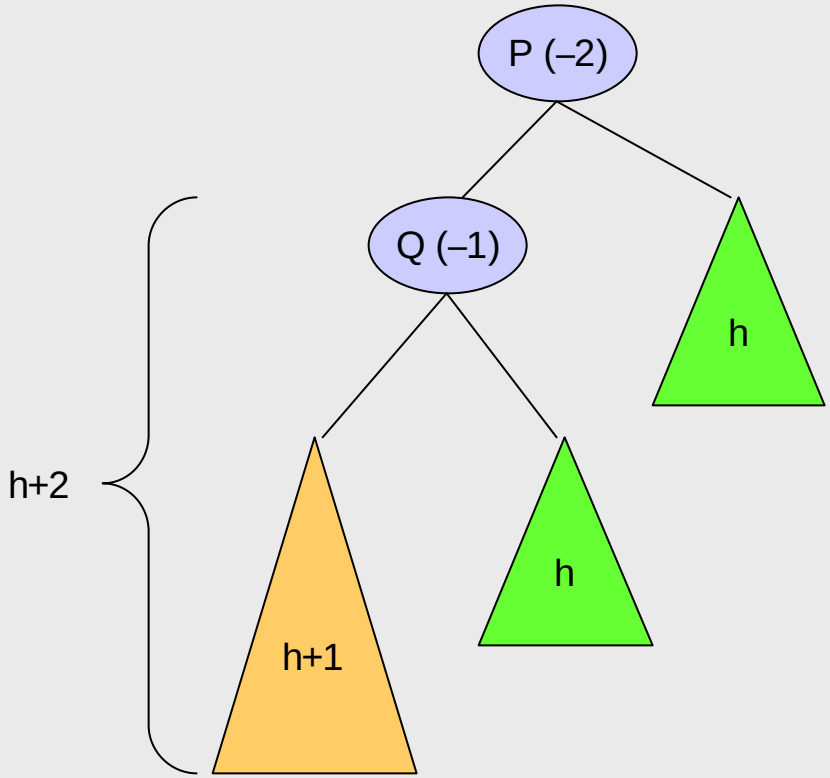


AVL tree after insertion into left subtree of left child Q

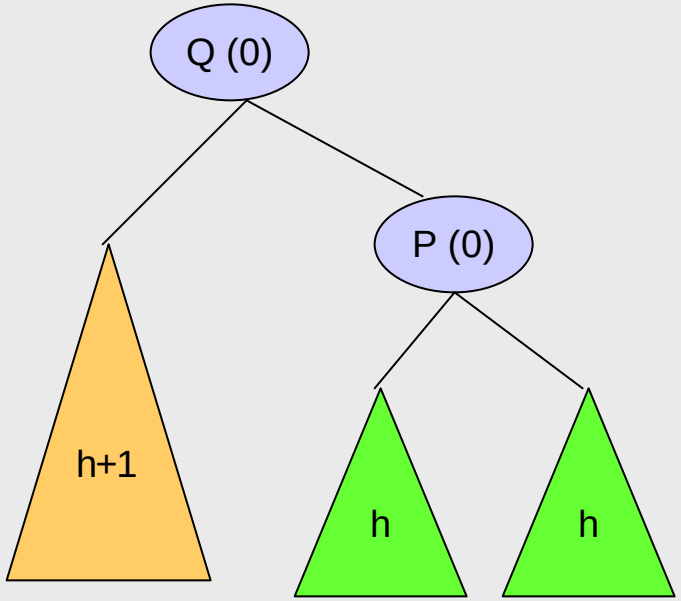
Handling the Imbalance for Symmetric Case 1a

- As the diagram on the previous slide clearly indicates, the AVL tree has become imbalanced due to the insertion of a new node into the left subtree of the left child of P.
- Note that the point of imbalance is at node P, the parent of the left child into whose subtree the insertion occurred.
- As with case 1, the imbalance is removed via a rotation of Q about P. In this case, however, the rotation will be a right rotation, effectively bringing the subtree rooted at Q up and to the right in the tree pushing P down and to the right. This is illustrated in the next slide.

Single Right Rotation Handles Case 1a Imbalance

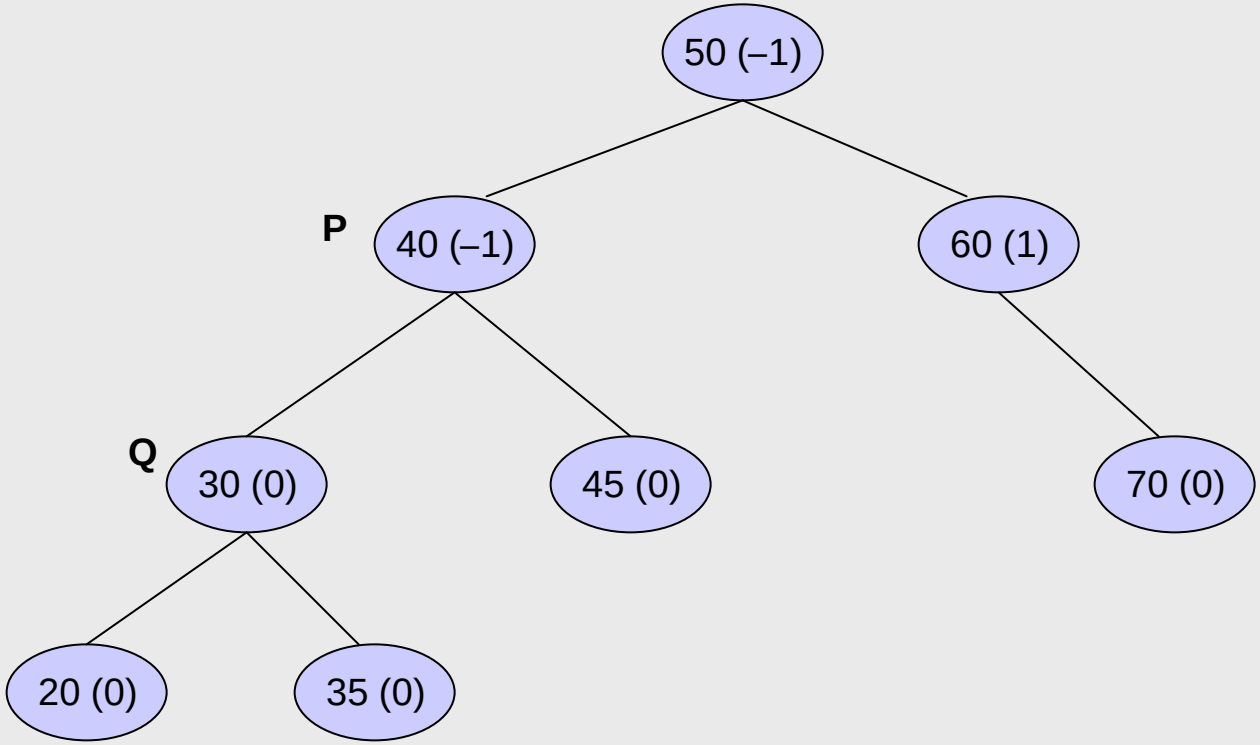


Result of a case 1a imbalance in an AVL tree caused by the insertion of a new node into the left subtree of the left child of P



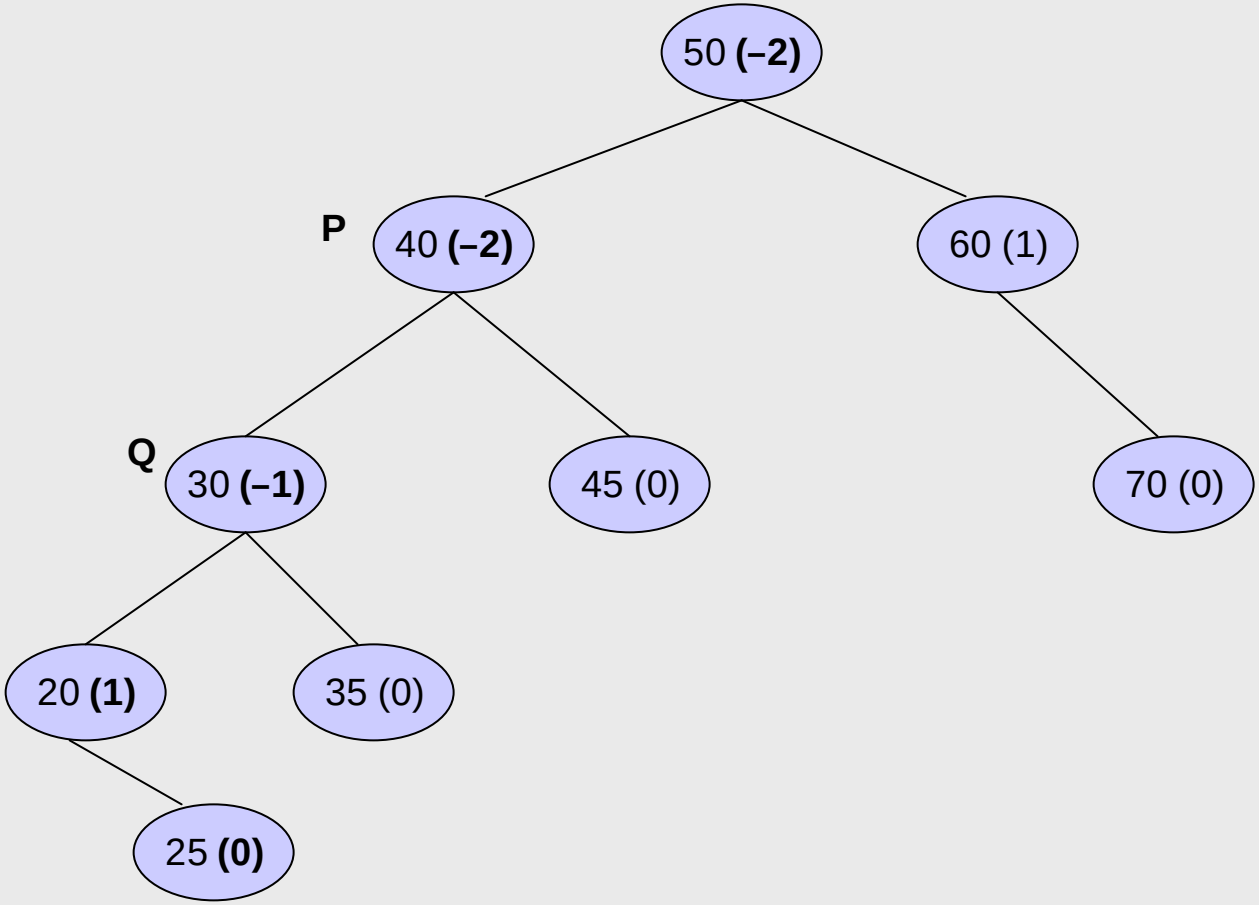
AVL tree after a single right rotation of Q about P. Balance has been restored to the tree.

Example – Case 1a Insertion Imbalance



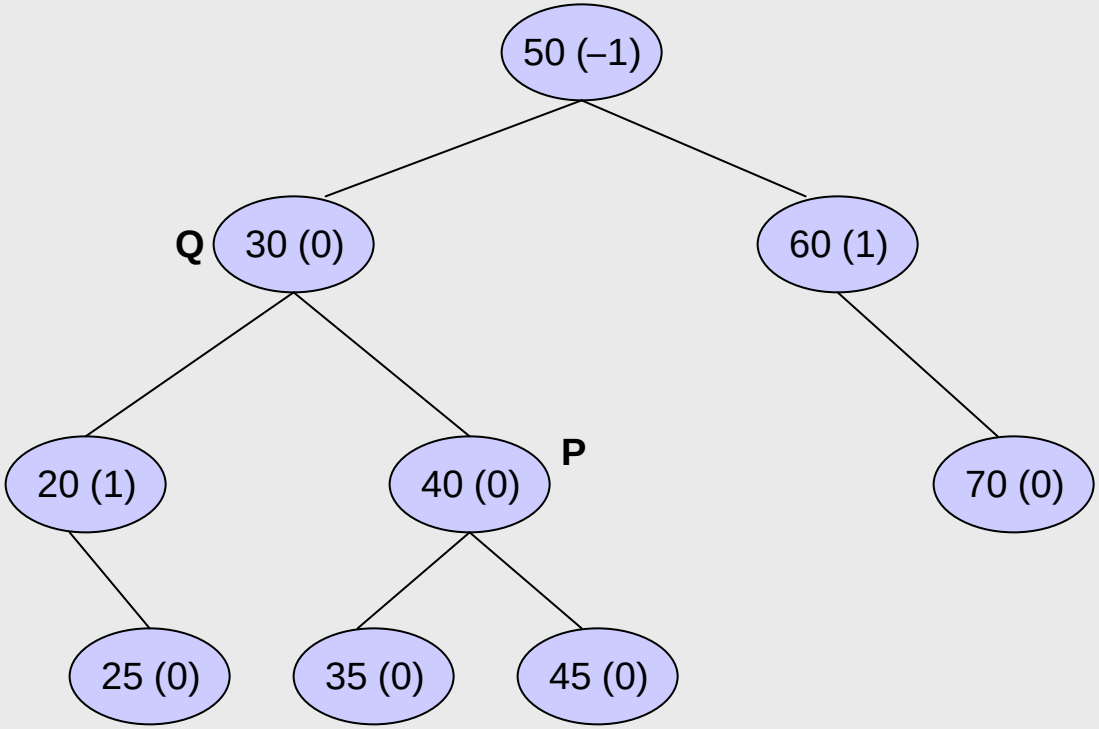
Initial AVL Tree

Example – Case 1a Insertion Imbalance



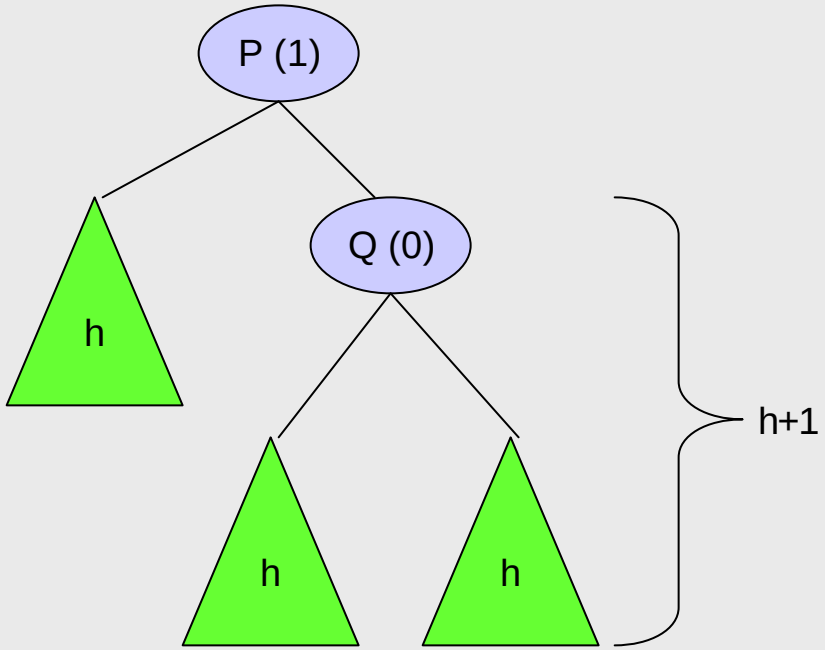
AVL tree after insertion of node with value 25. Insertion in the left subtree of the left child P. Balance factors that have changed from the initial tree are shown in bold.

Example – Case 1a Insertion Imbalance

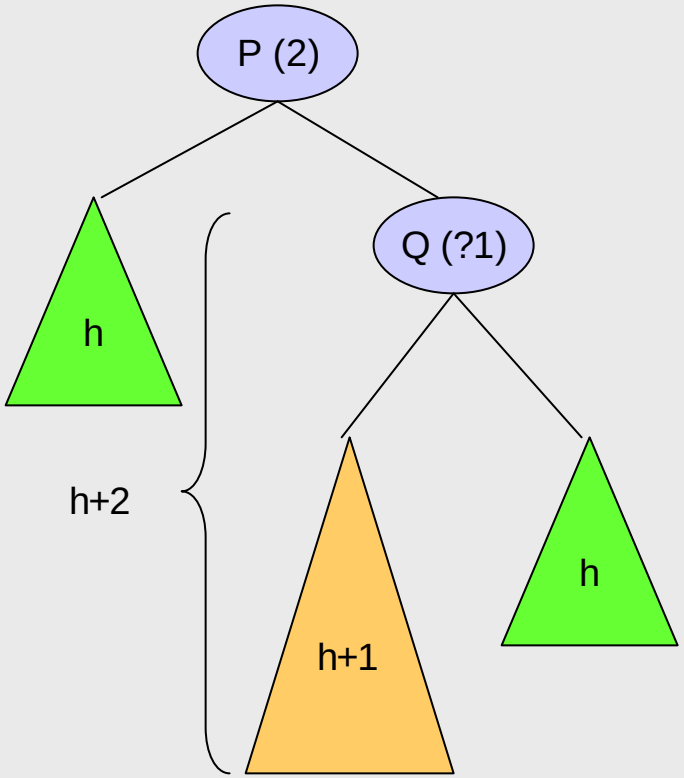


AVL tree after a single right rotation of 30 about 40. Note that this balances the tree all the way to the root.

Case 2: Insertion Into the Left Subtree of a Right Child

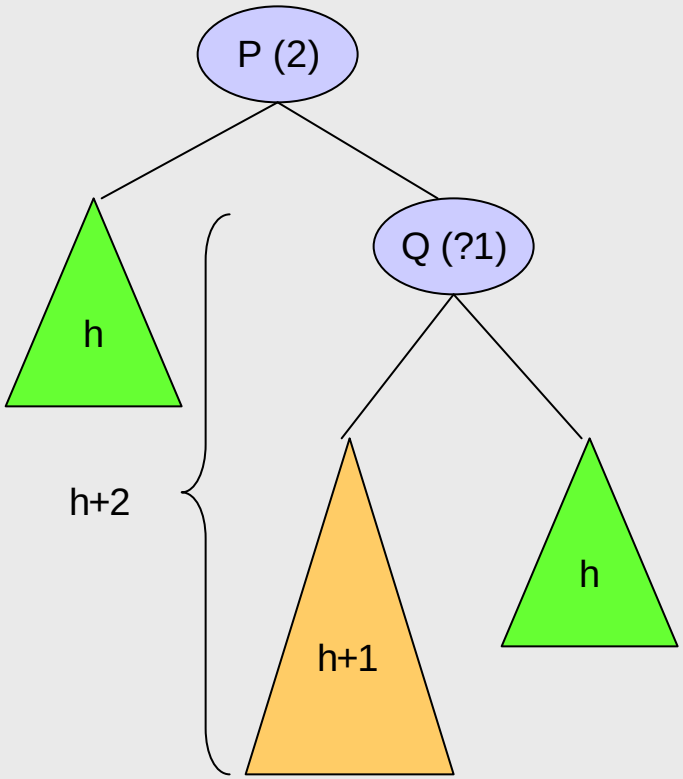


Initial AVL tree before insertion

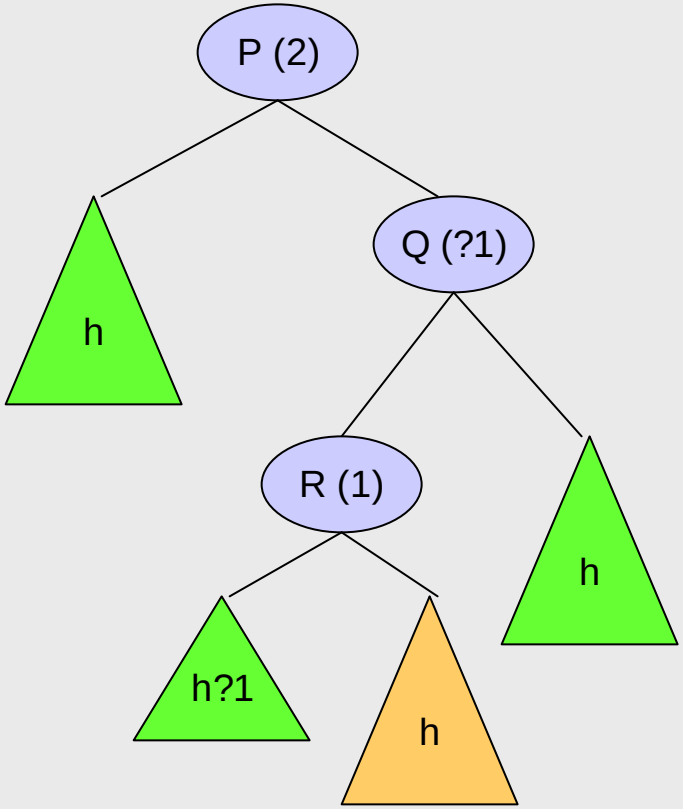


AVL tree after insertion into
left subtree of right child Q

A Closer Look At What Happens To The Tree In Case 2



AVL tree after insertion into left subtree of right child Q

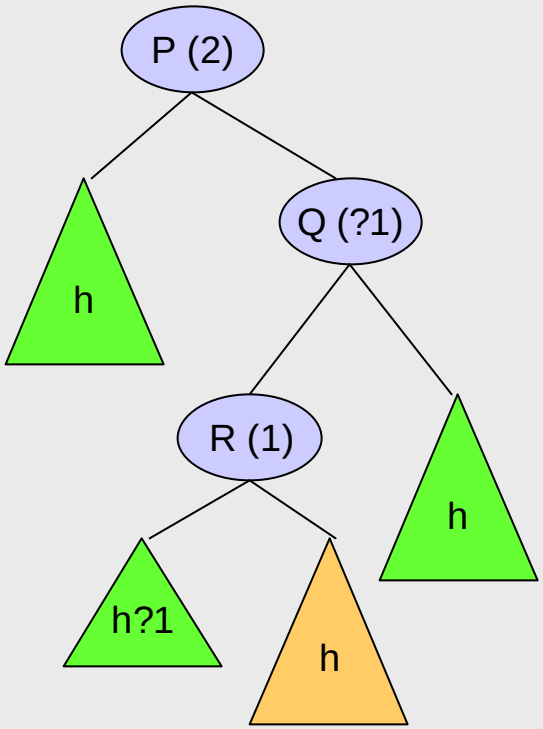


Assuming insertion was in the right subtree of the left child of Q (call this node R)

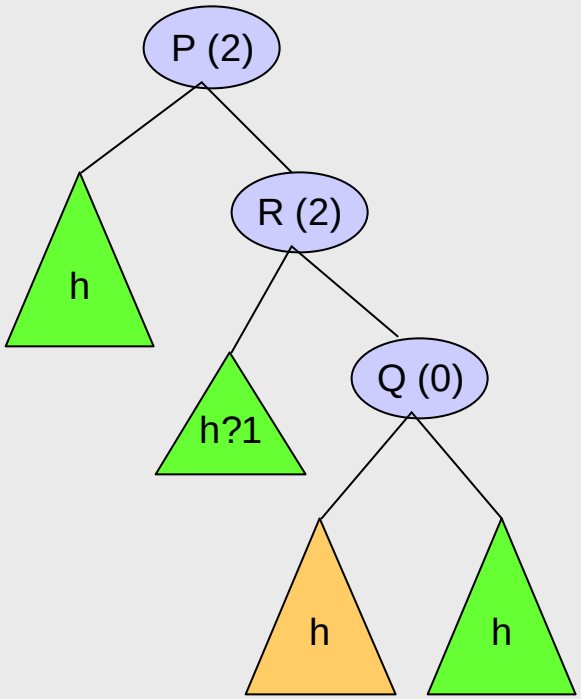
Handling the Imbalance for Case 2

- As the diagram on the previous slide clearly indicates, the AVL tree has become imbalanced due to the insertion of a new node into the left subtree of the right child of P.
- As can be seen on the next slide, a single rotation does not correct the imbalance of the tree rooted at P (it actually induces a further imbalance in the tree rooted at R after the first rotation occurs).
- The solution to the problem is a second rotation. For case 2, (insertion was in the left subtree of a right child) the first rotation is a right rotation of R about Q and the second rotation is a left rotation of R about P. For the symmetric case 2a (insertion is in the right subtree of a left child), we'll see shortly that the first rotation will be a right rotation and the second will be a left rotation.

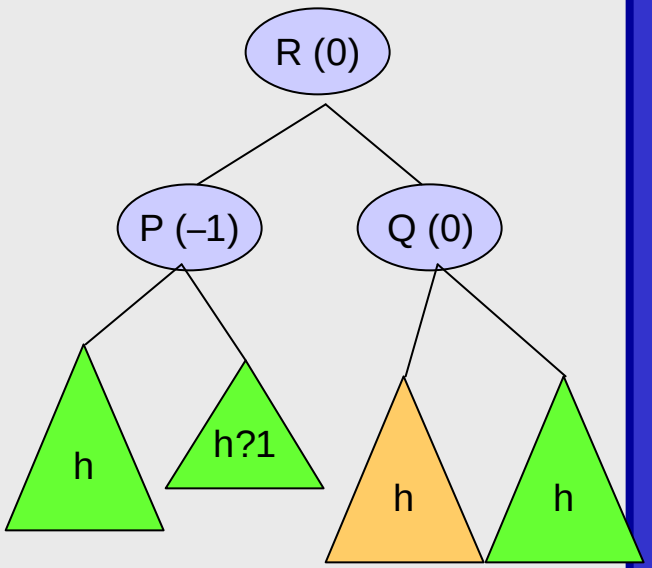
Double Rotation Handles Case 2 Imbalance



Resulting imbalance in case 2 insertion

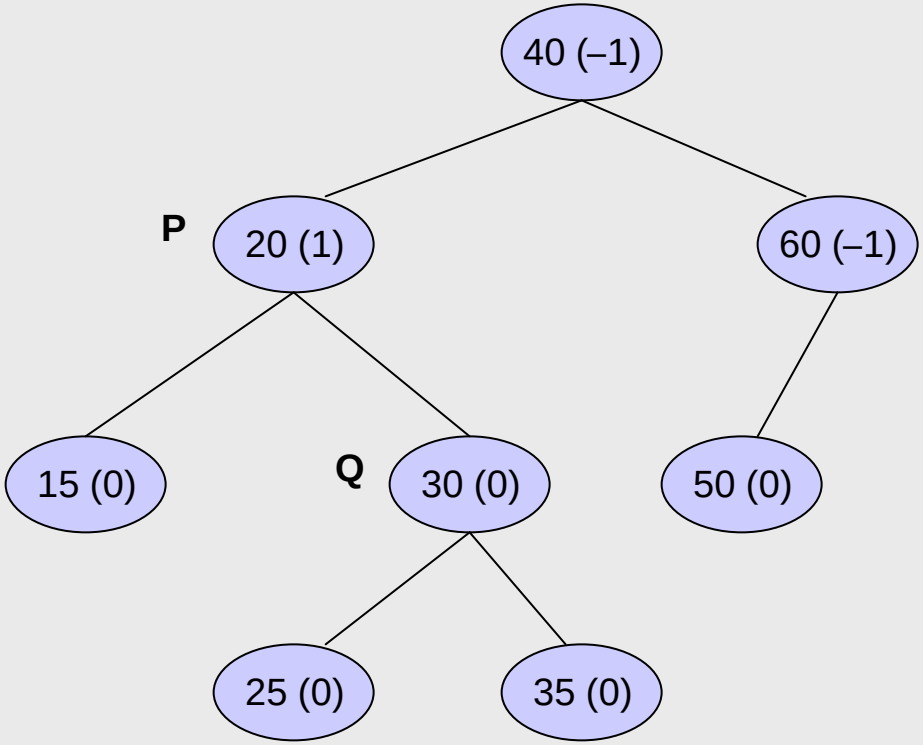


After right rotation of R about Q



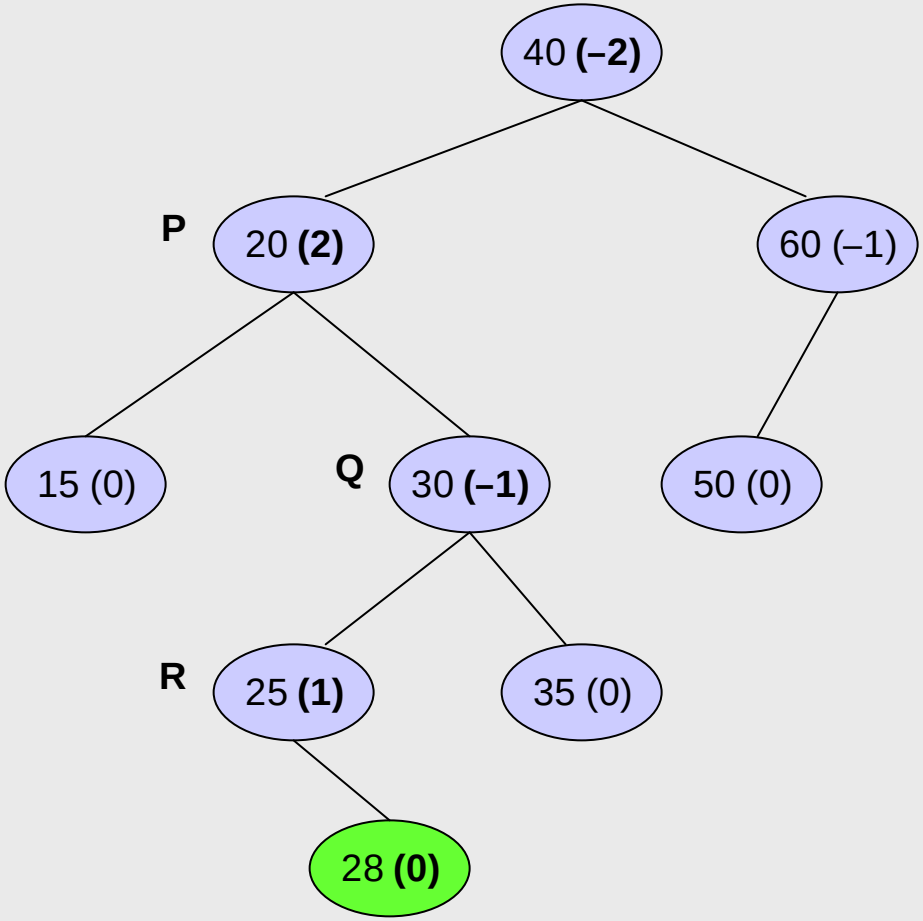
After left rotation of R about P. The complete set of rotations has been a right followed by a left or a RL double rotation.

Example – Case 2 Insertion Imbalance



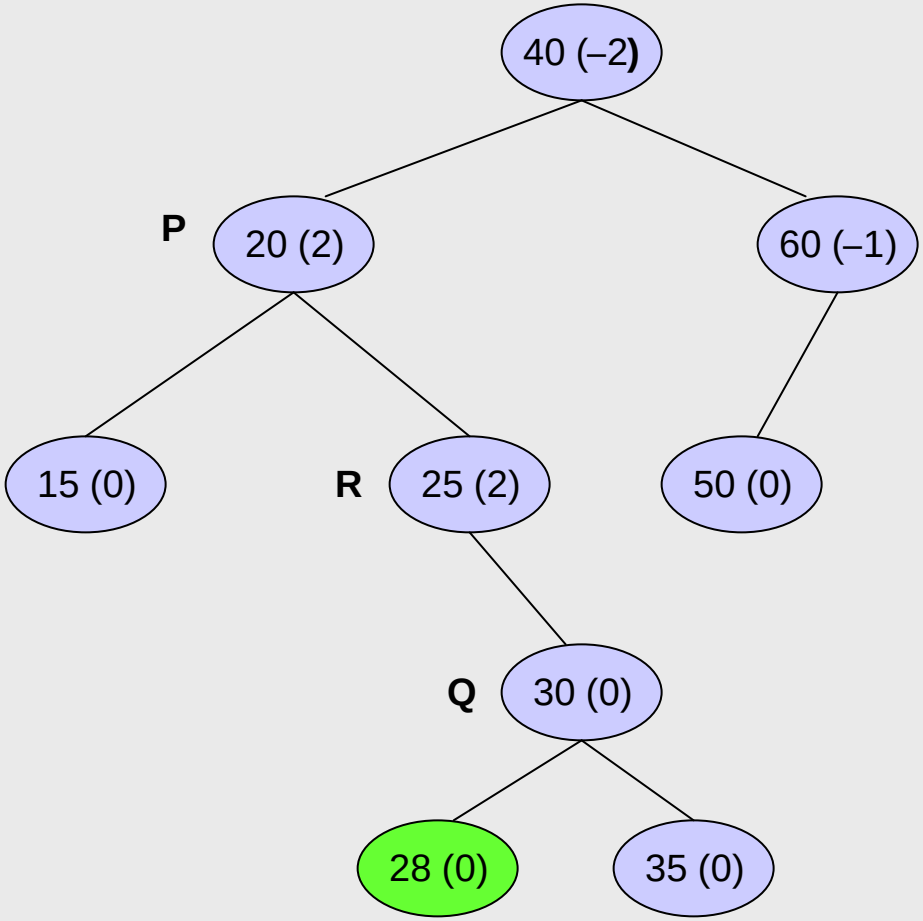
Initial AVL Tree

Example – Case 2 Insertion Imbalance



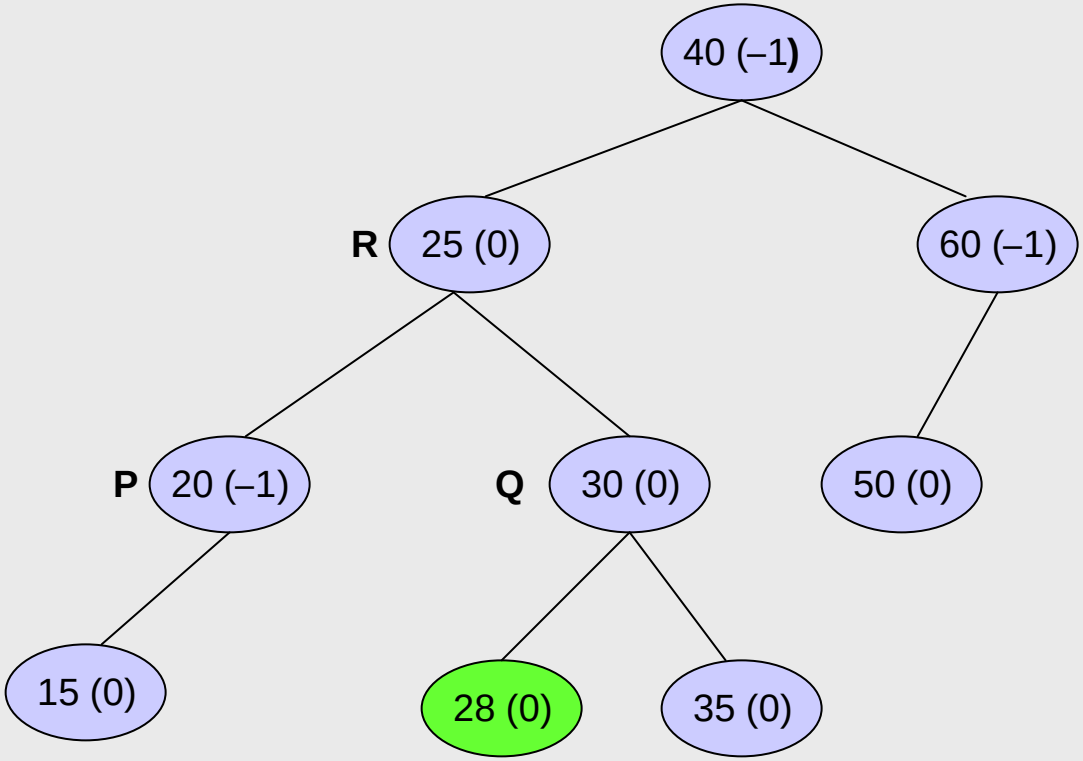
AVL tree after insertion of node with value 28. Balance factors that have changed from the initial tree are shown in bold.

Example – Case 2 Insertion Imbalance



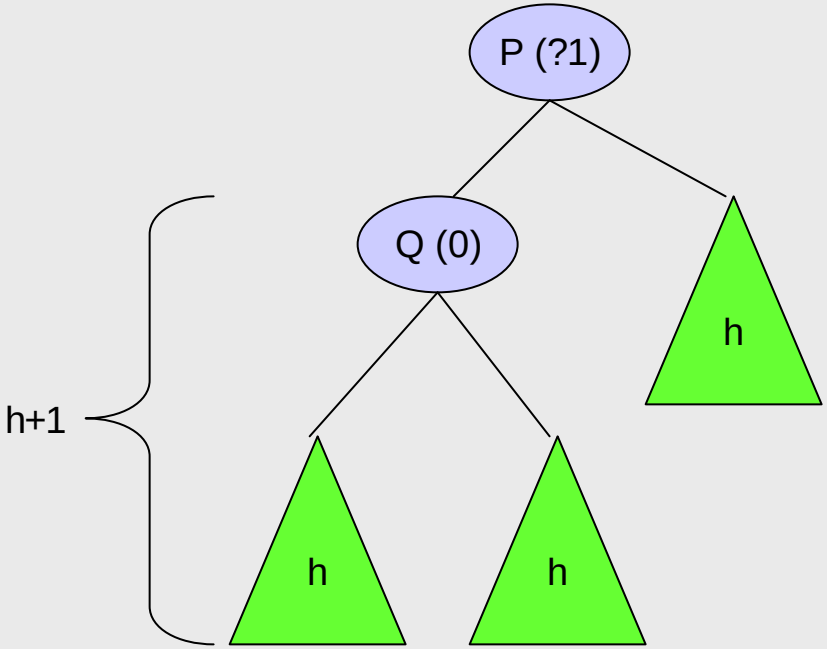
AVL tree after right rotation of R about Q (25 about 30). Note that the tree is not balanced after this first rotation.

Example – Case 2 Insertion Imbalance

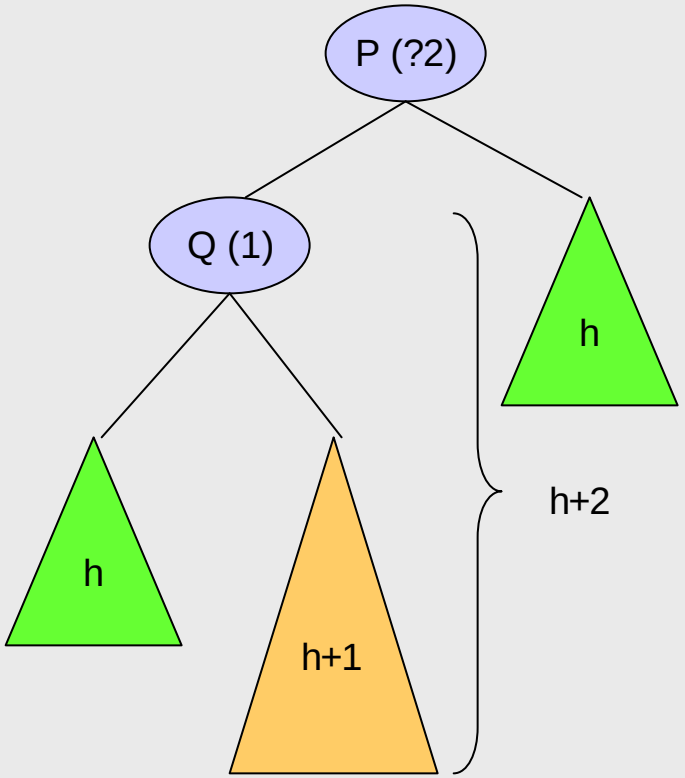


AVL tree after a second rotation, this one a left rotation of R about P (25 about 20). Note that the tree is now balanced.

Symmetric Case 2a: Insertion Into the Right Subtree of a Left Child

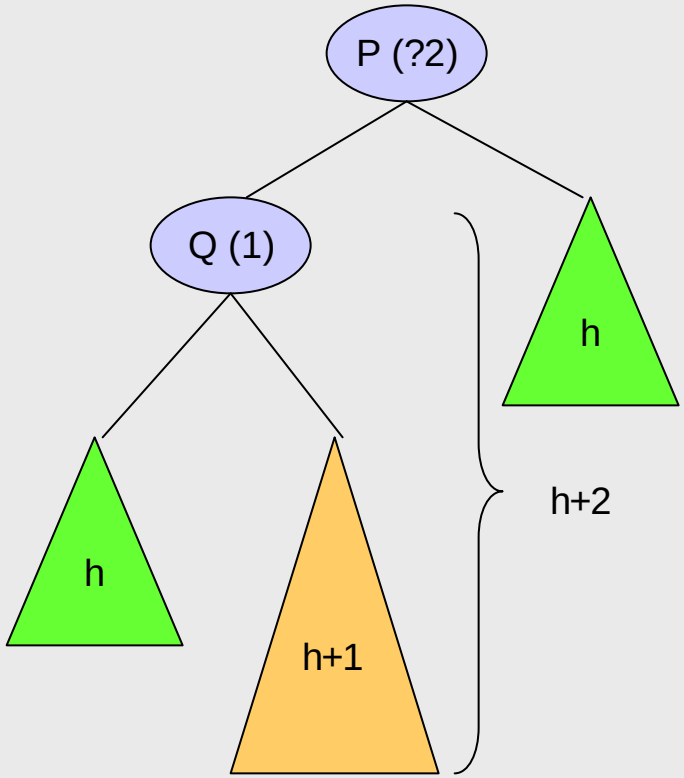


Initial AVL tree before insertion

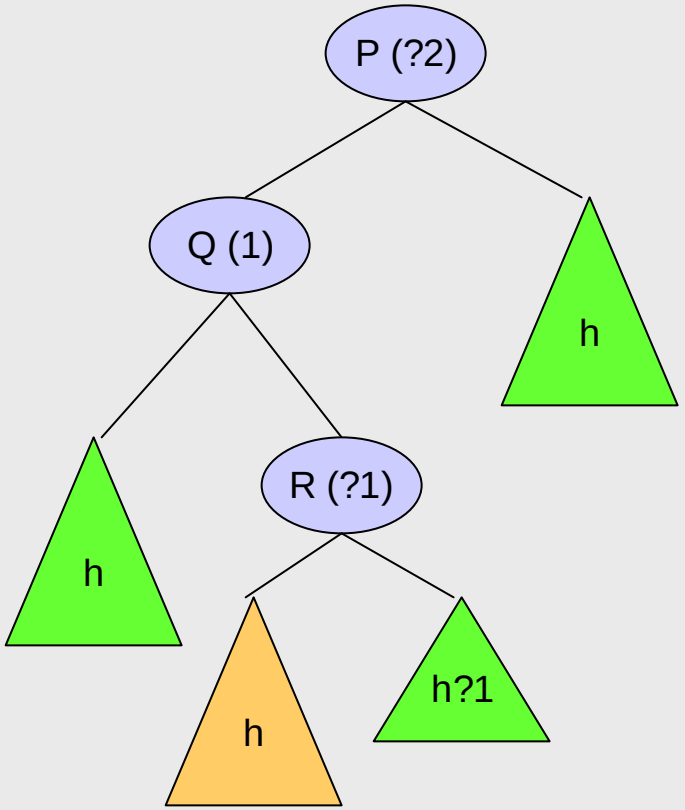


AVL tree after insertion into right subtree of left child Q

A Closer Look At What Happens To The Tree In Symmetric Case 2a



AVL tree after insertion into right subtree of left child Q

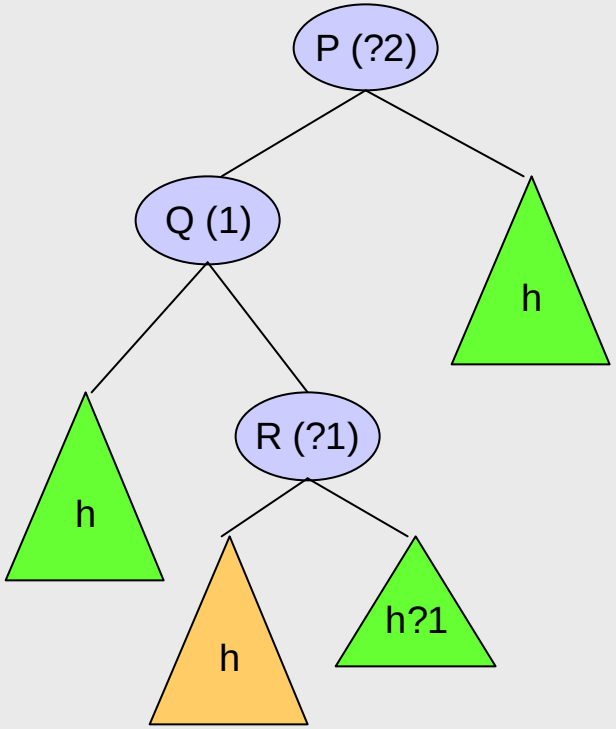


Assuming insertion was in the left subtree of the right child of Q (call this node R)

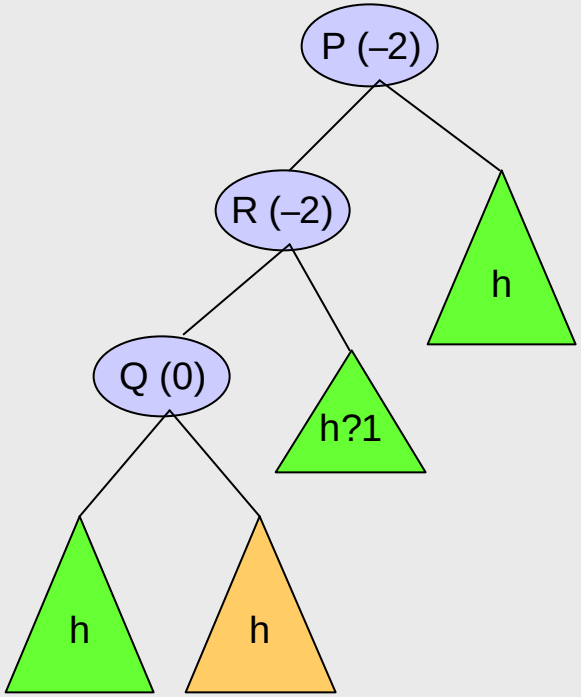
Handling the Imbalance for Symmetric Case 2a

- As the diagram on the previous slide clearly indicates, the AVL tree has become imbalanced due to the insertion of a new node into the right subtree of the left child of P.
- As was the case for the symmetric case 2, a double rotation is required to remove the imbalance in the tree. In this case, the first rotation will be a left rotation of R about Q followed by a right rotation of R about P.
- This is illustrated in the next slide, followed by an example.

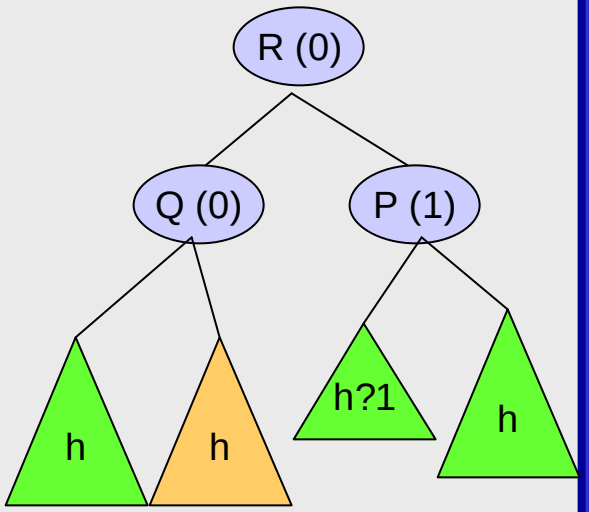
Double Rotation Handles Symmetric Case 2a Imbalance



Symmetric case 2a insertion imbalance.

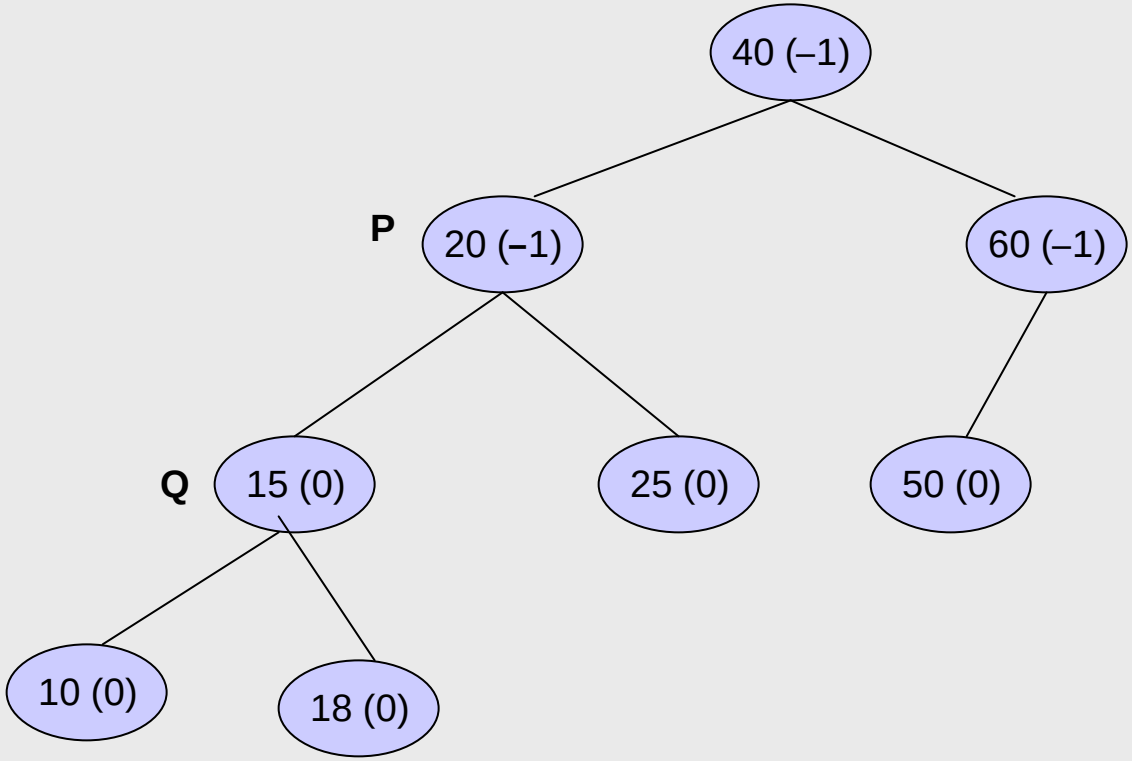


After left rotation of R about Q. Note tree is still imbalanced, now both at R and P.



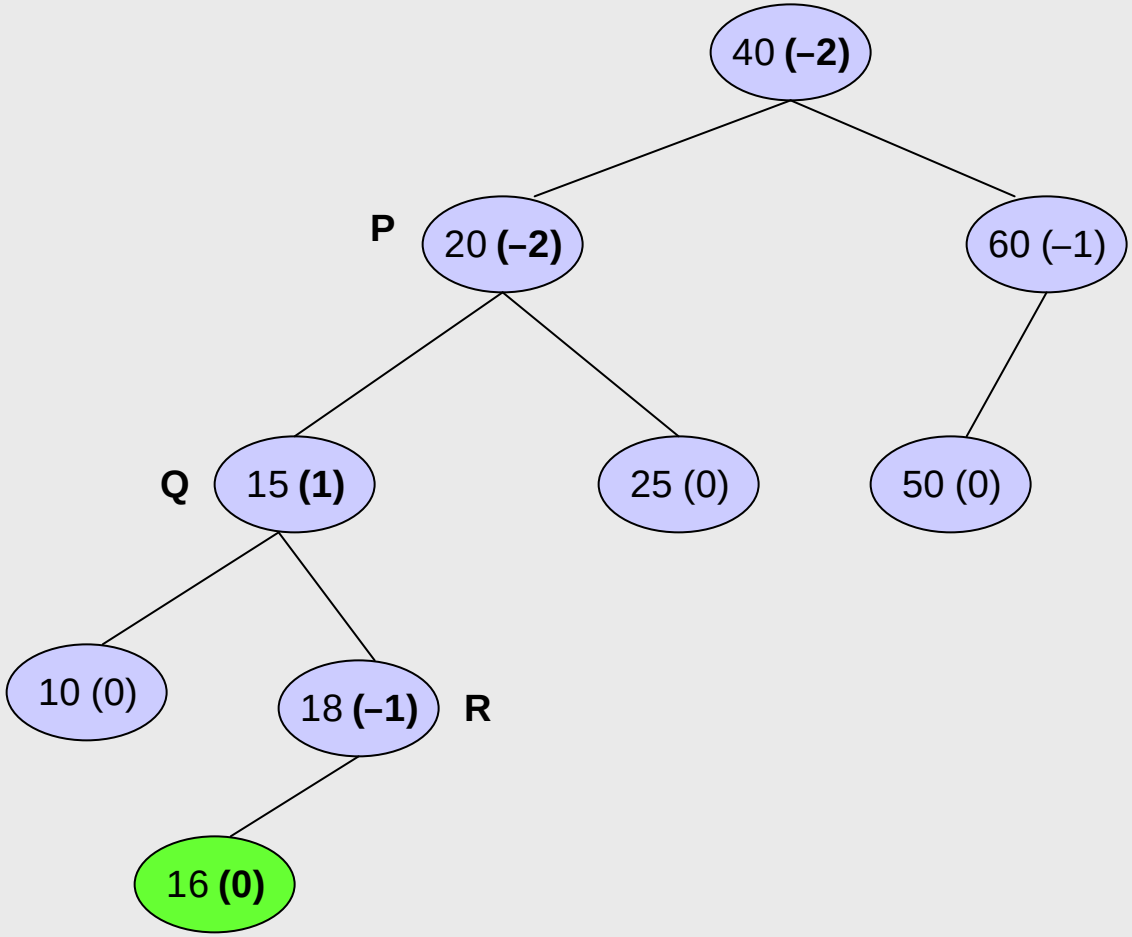
After right rotation of Q about R. Note tree is now balanced.

Example – Symmetric Case 2a Insertion Imbalance



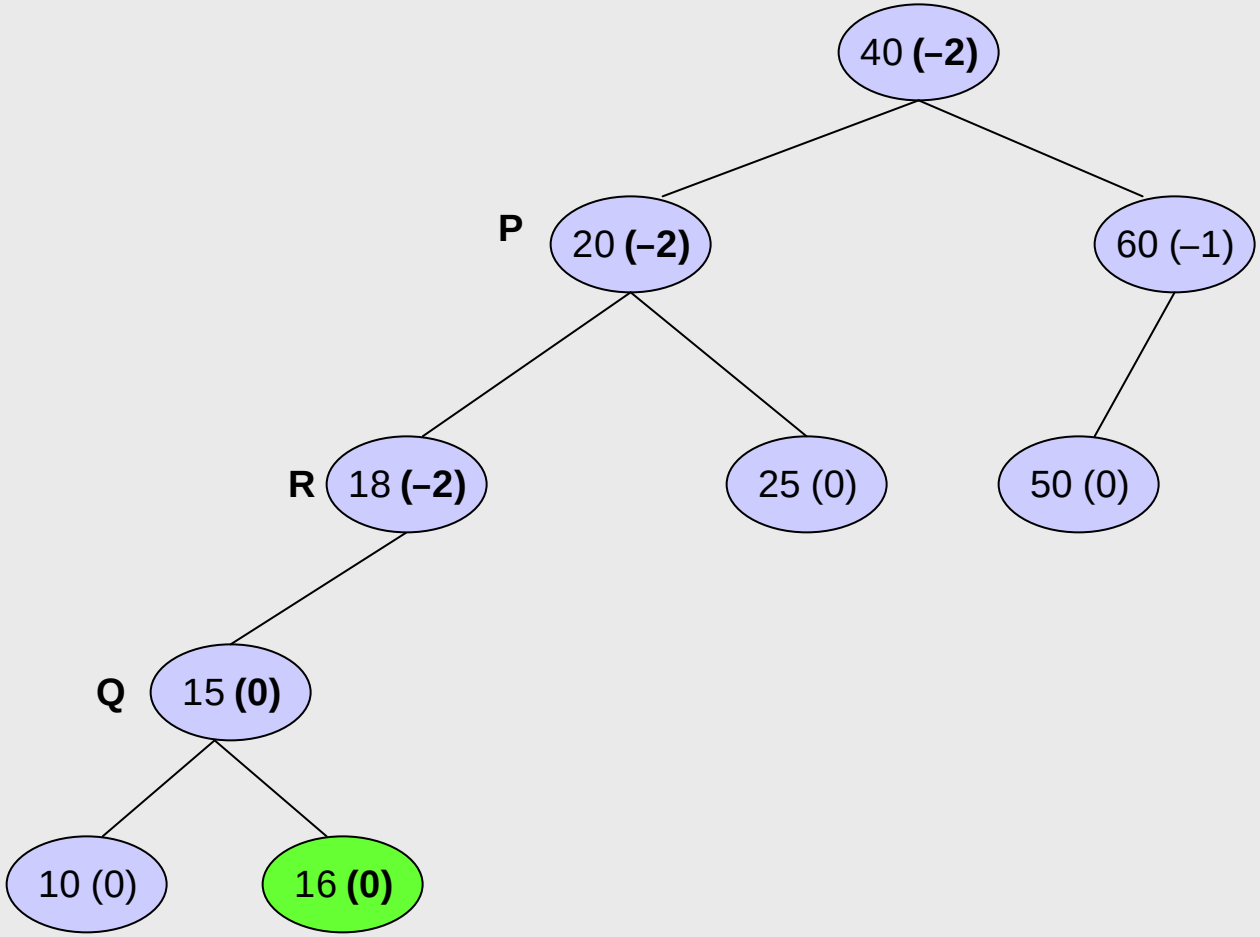
Initial AVL Tree

Example – Symmetric Case 2a Insertion Imbalance



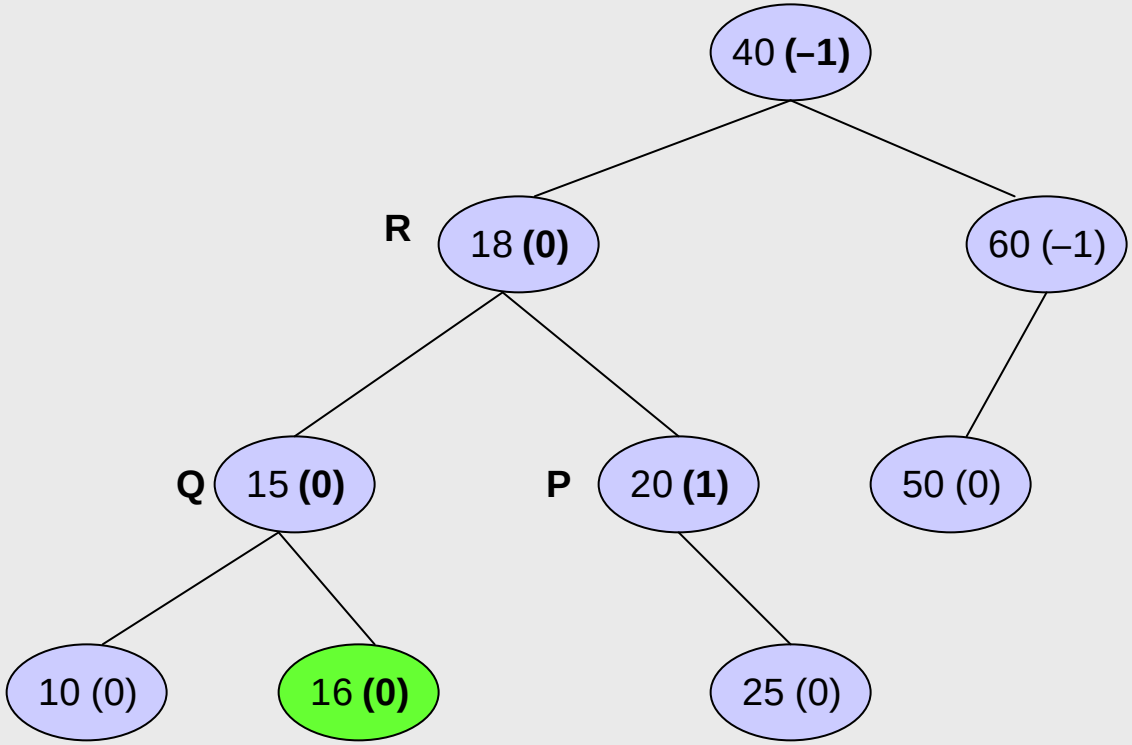
AVL tree after insertion of node with value 16. Balance factors that have changed are shown in bold.

Example – Symmetric Case 2a Insertion Imbalance



AVL tree after left rotation of 18 about 15 (R about Q). Notice that the balance factors clearly indicate the tree is not balanced.

Example – Symmetric Case 2a Insertion Imbalance



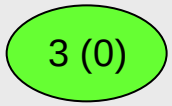
AVL tree after right rotation of 18 about 10 (R about P). Tree is now balanced.

One Big Example of AVL Tree Insertions

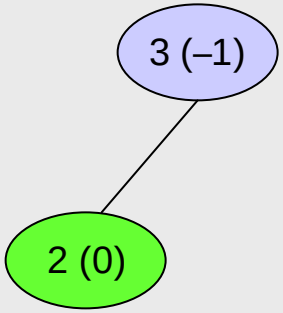
- Over the next few slides we'll work one big example constructing an AVL tree and show the balancing that must occur due to each insertion.
- Suppose that we start with an initially empty AVL tree and insert values in the following order:

  3, 2, 1, 4, 5, 6, 7, 16, 15, 14, 13, 12, 11, 10, 8, 9
- Try constructing this tree yourself, then look at the steps on the following pages to check your work.

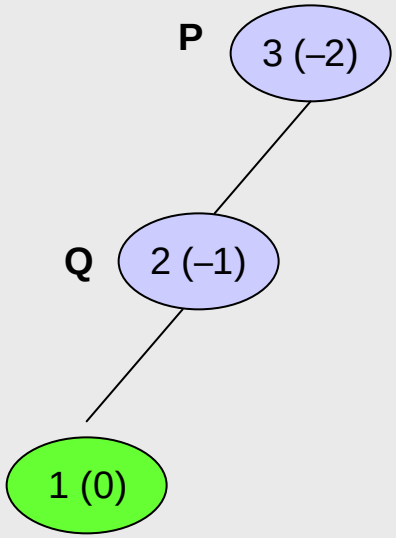
Big Example – Insert 3, Insert 2, Insert 1



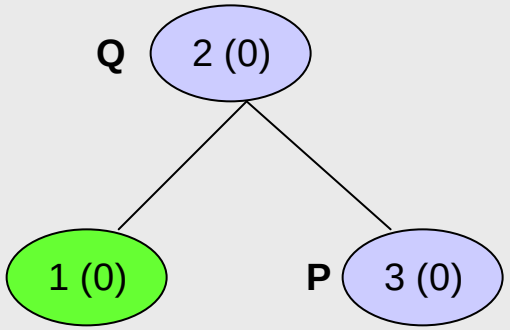
Step 1
insert 3



Step 2
insert 2



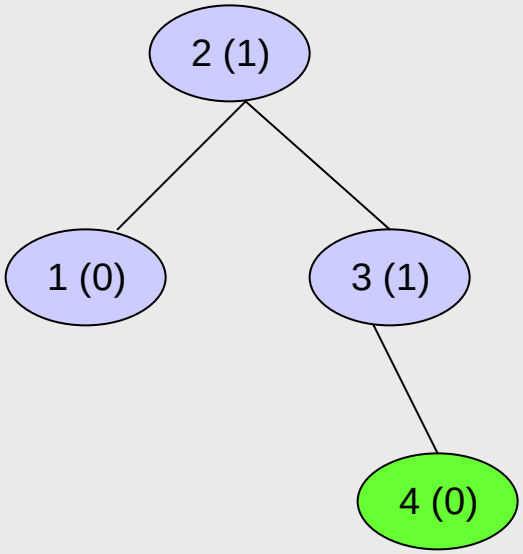
After insertion



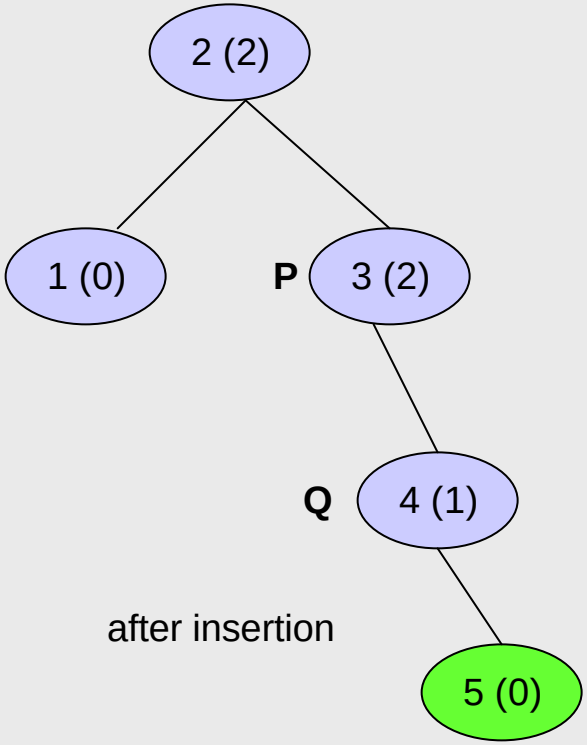
After right rotation of
Q about P

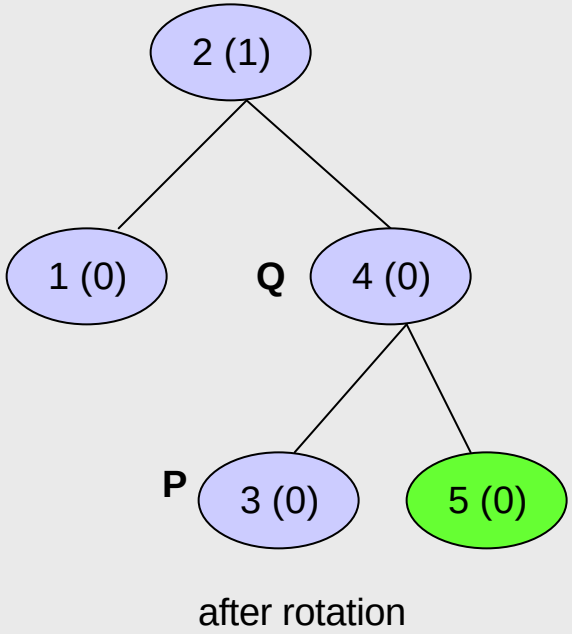
Step 3: insert 3
Case 1a insertion

Big Example – Insert 4, Insert 5



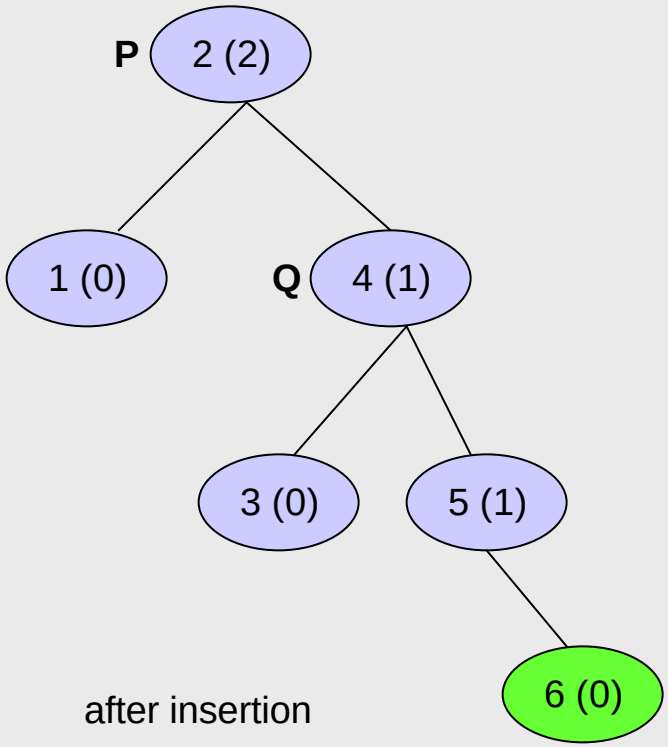
Step 4
insert 4



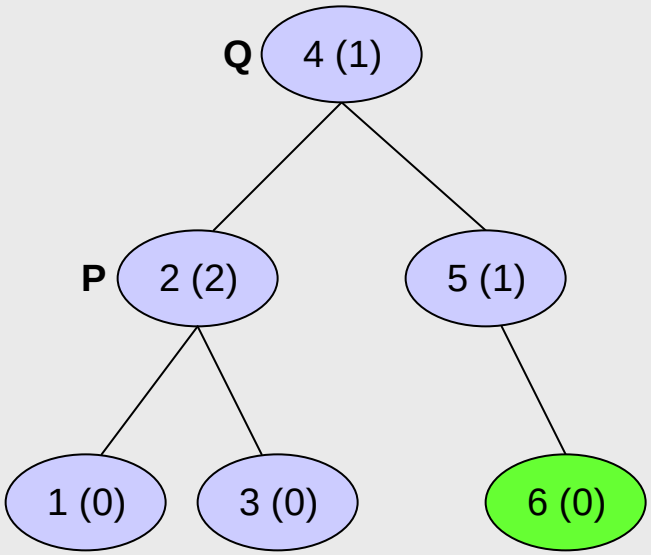


Step 5: insert 5
Case 1 insertion

Big Example – Insert 6



after insertion

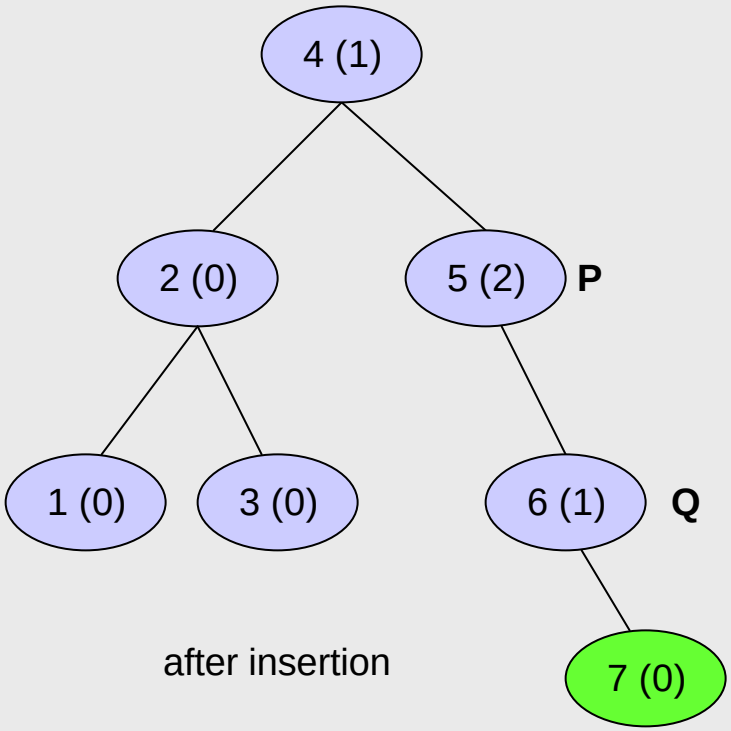


after rotation

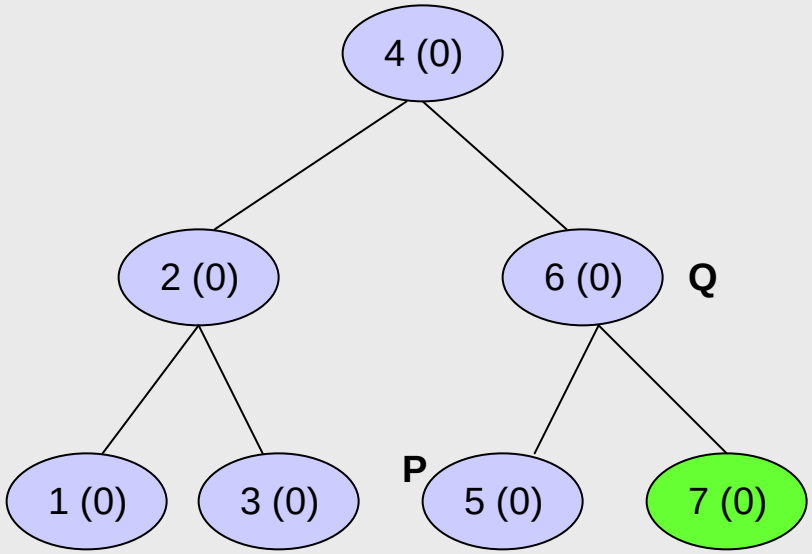
Step 6: insert 6
Imbalance occurs at root

Insertion occurred in the right subtree of a right child, thus a left rotation is required. The imbalance occurs at the root node of the tree in this case, so the left rotation is of 4 about 2. Recall that the node about which the rotation is occurring is labeled P and the node which is rotating about P labeled Q. So Q will be identified as P's right child.

Big Example – Insert 7



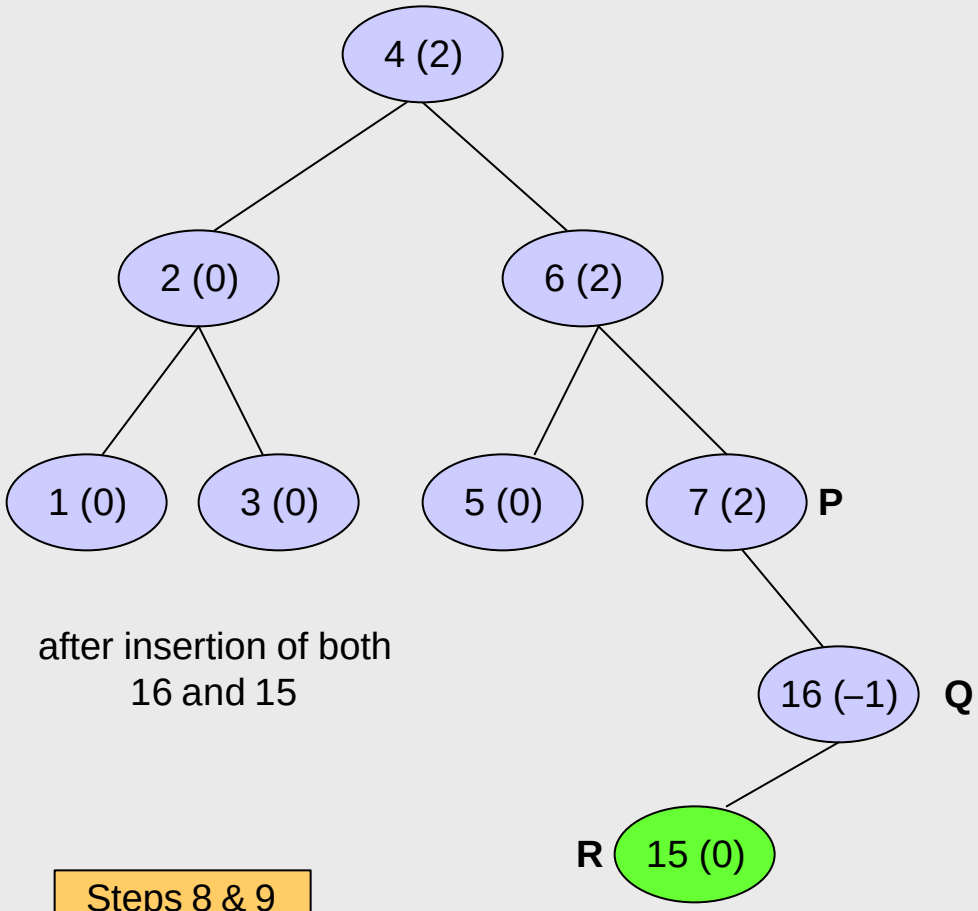
after insertion



after rotation

Step 7: insert 7
Case 1a insertion

Big Example – Insert 16, Insert 15

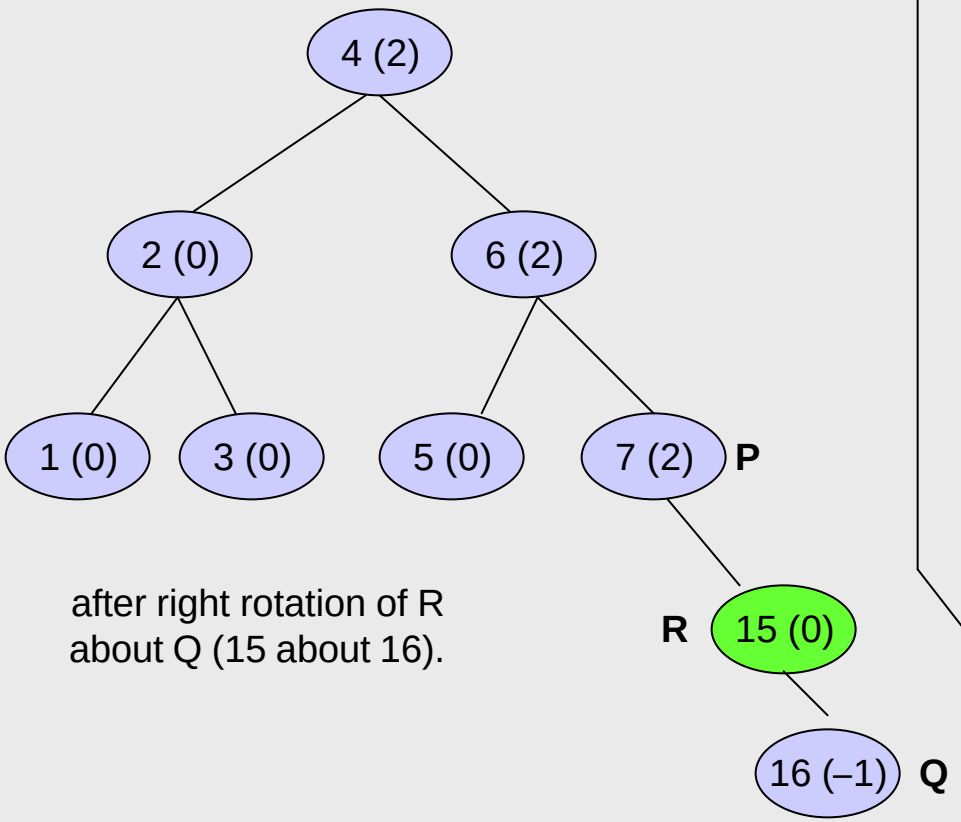


after insertion of both 16 and 15

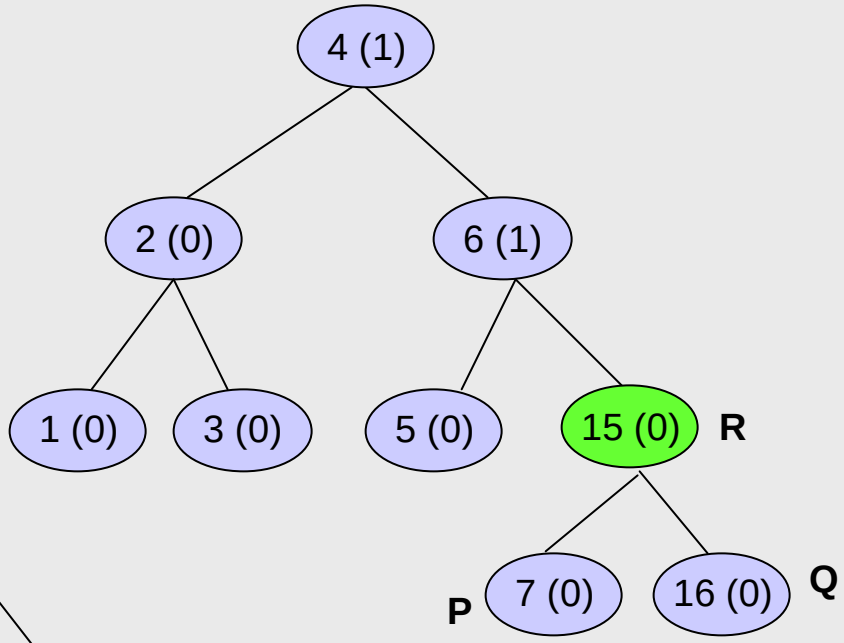
Steps 8 & 9
insert 16
insert 15

The insertion of 16 causes no imbalance, however, the insertion of 15 causes an imbalance at node 7. Node 7 is designated as P with its right child Q being node 16. Thus, the insertion has occurred in the left subtree of a right child which is a case 2 insertion. A double right-left rotation is required to rebalance the tree.

Big Example – Right-Left Rotation After Insertion of 15

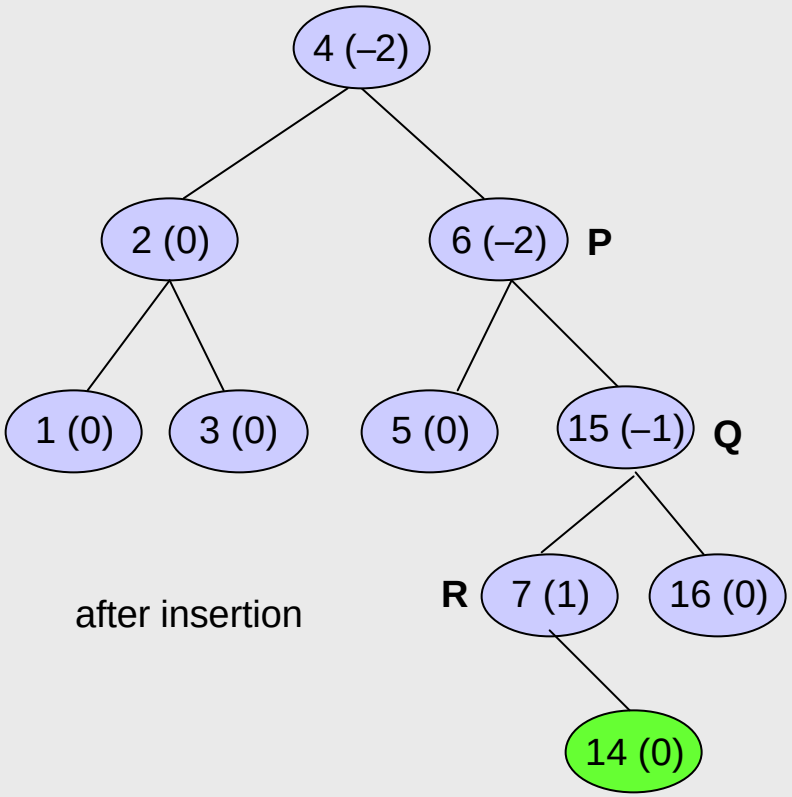


after right rotation of R about Q (15 about 16).



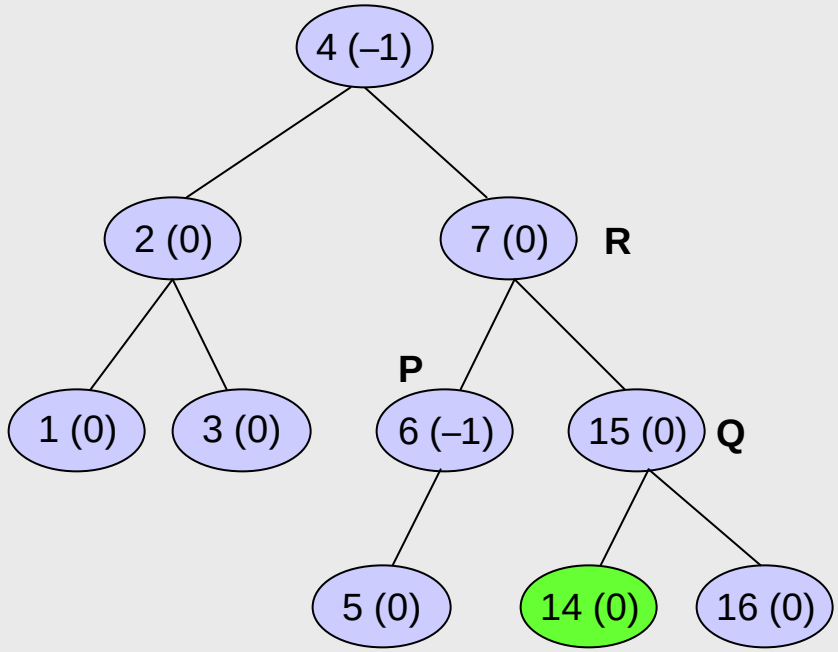
after left rotation of R about P (15 about 7).

Big Example – Insert 14



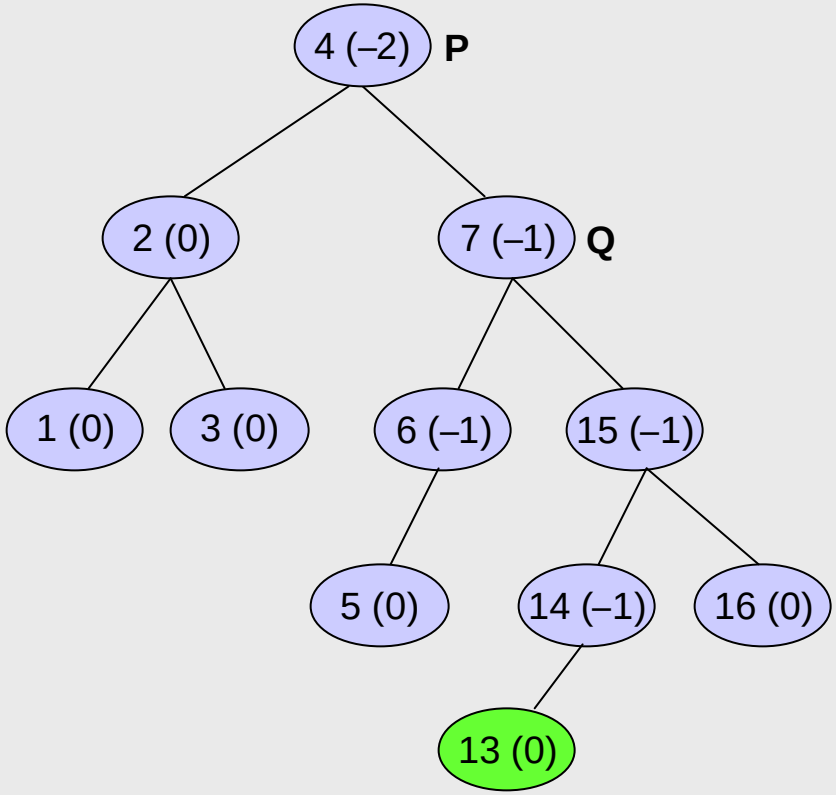
after insertion

Step 10
insert 14
Case 2 insertion

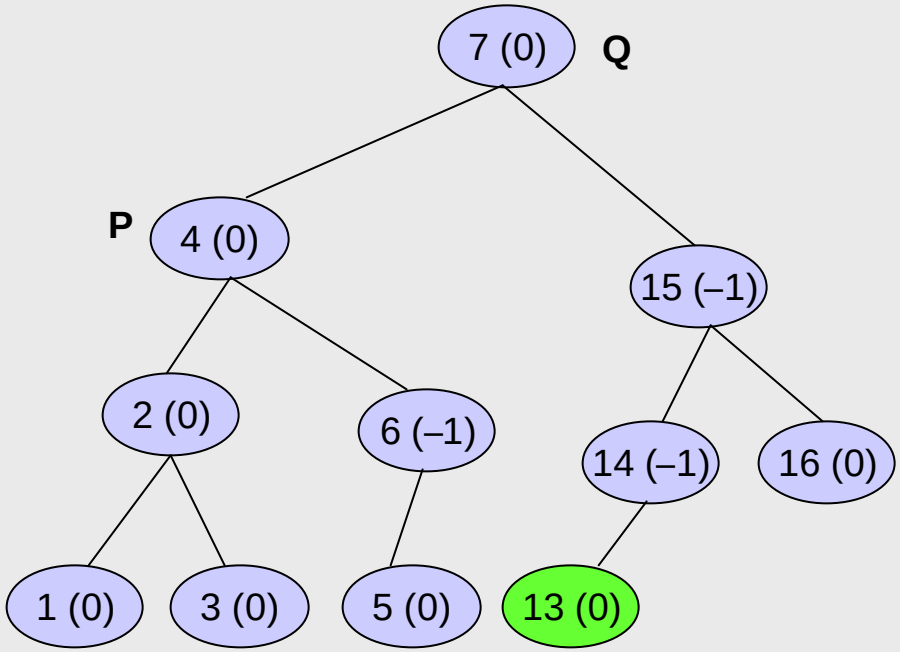


after right rotation of 7 about 15 followed by a left rotation of 7 about 6.

Big Example – Insert 13

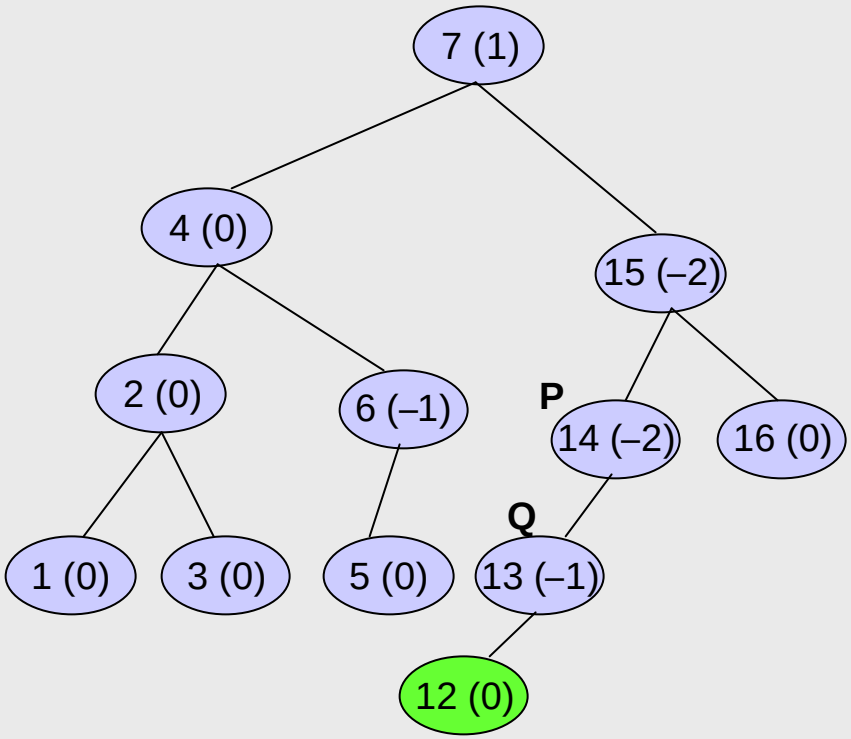


Step 11
insert 13
Case 1 insertion

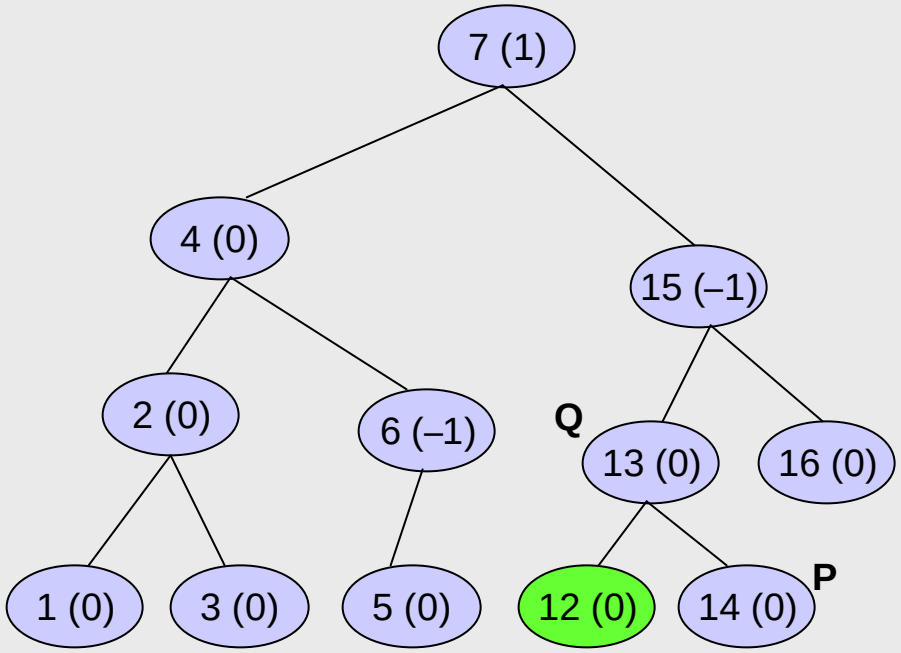


after single left rotation of 7 about 4.

Big Example – Insert 12

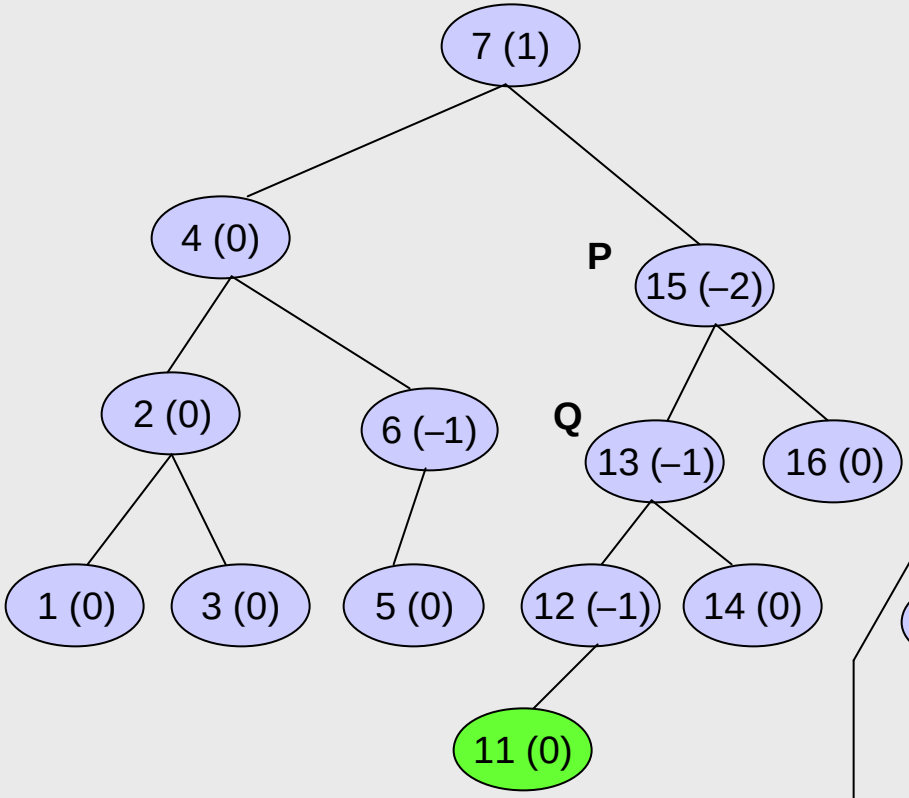


Step 12
insert 12
Case 1a insertion

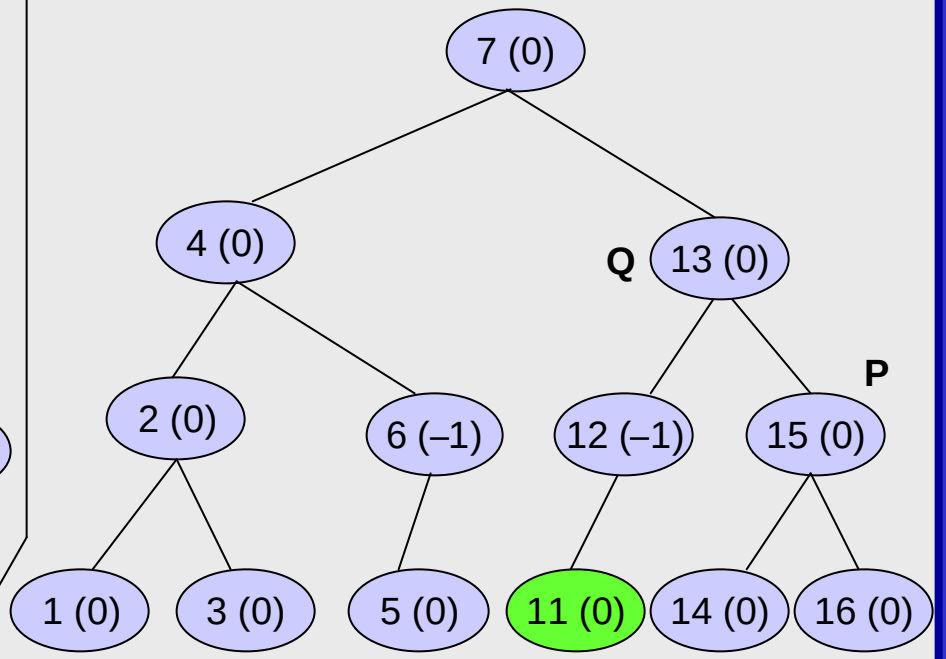


after single left rotation of 13 about 14.

Big Example – Insert 11

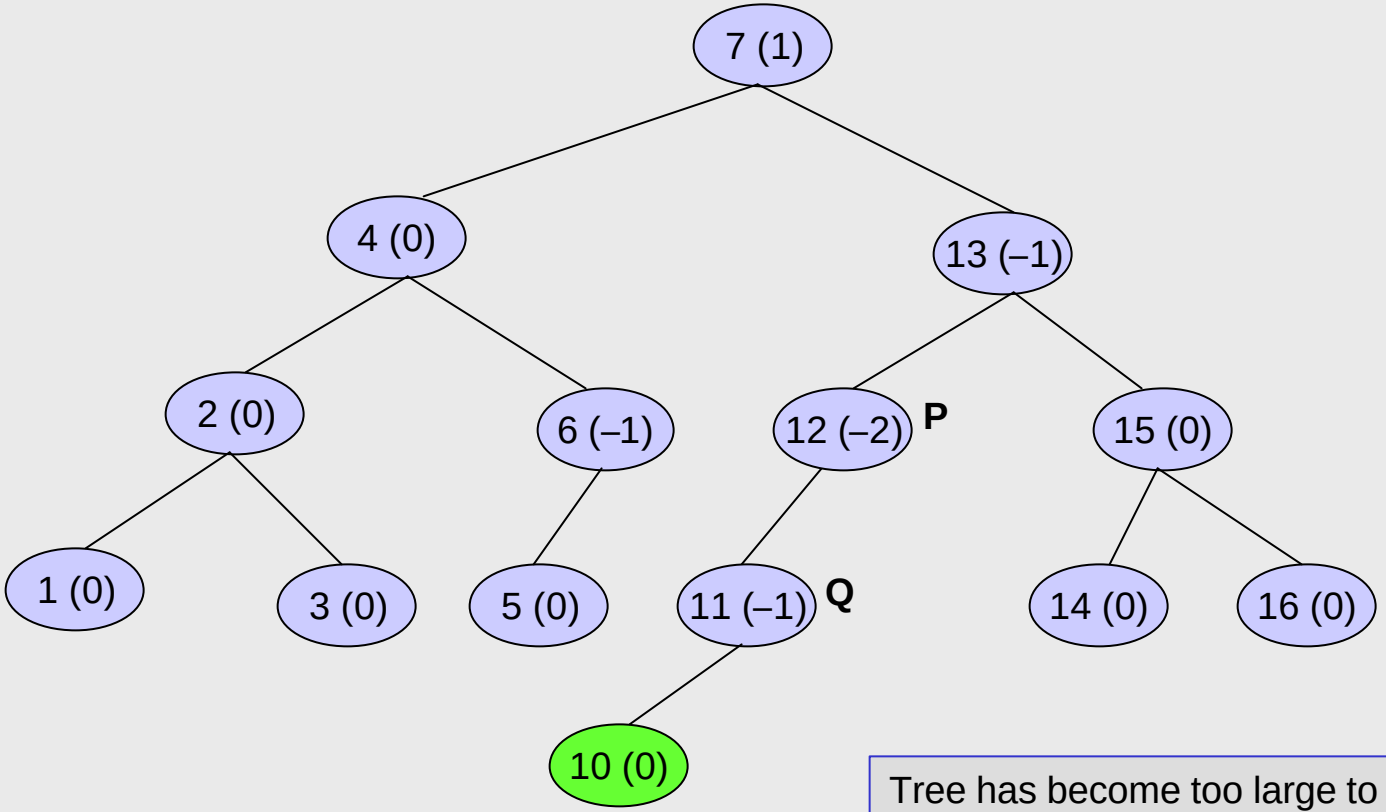


Step 13
insert 11
Case 1a insertion



after single left rotation of 13 about 15.

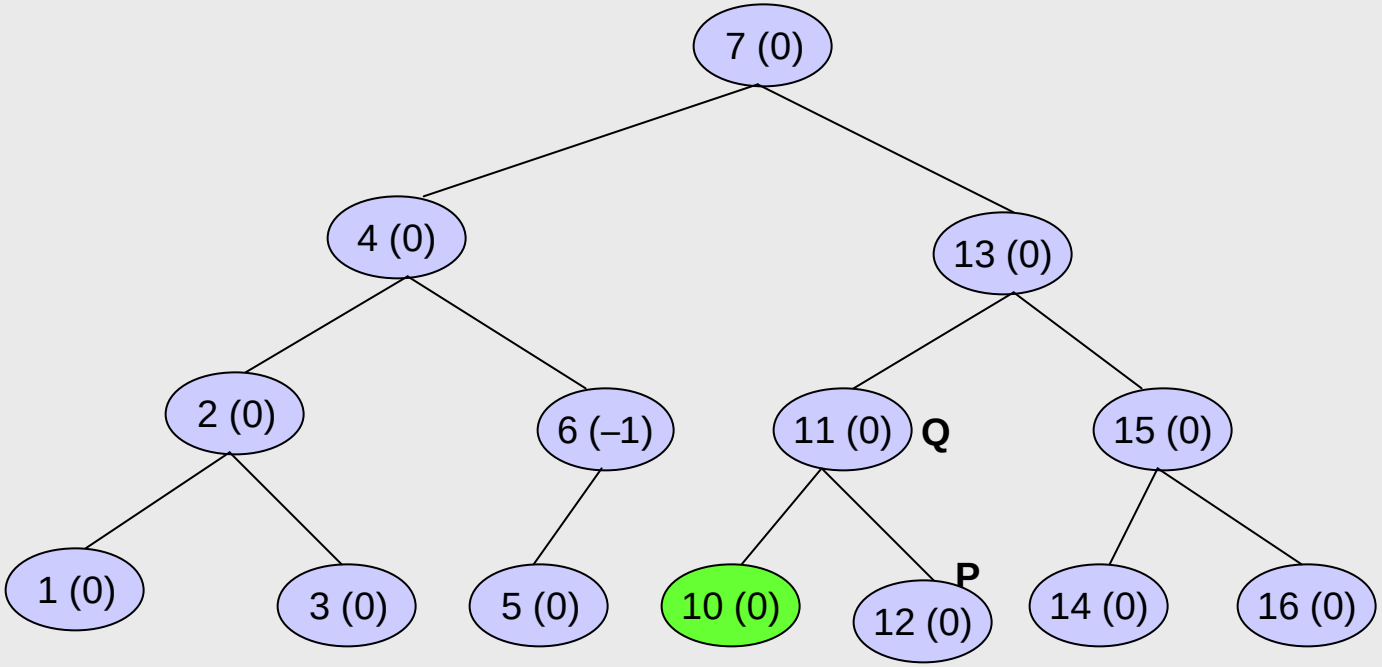
Big Example – Insert 10



Step 14
insert 10
Case 1a insertion

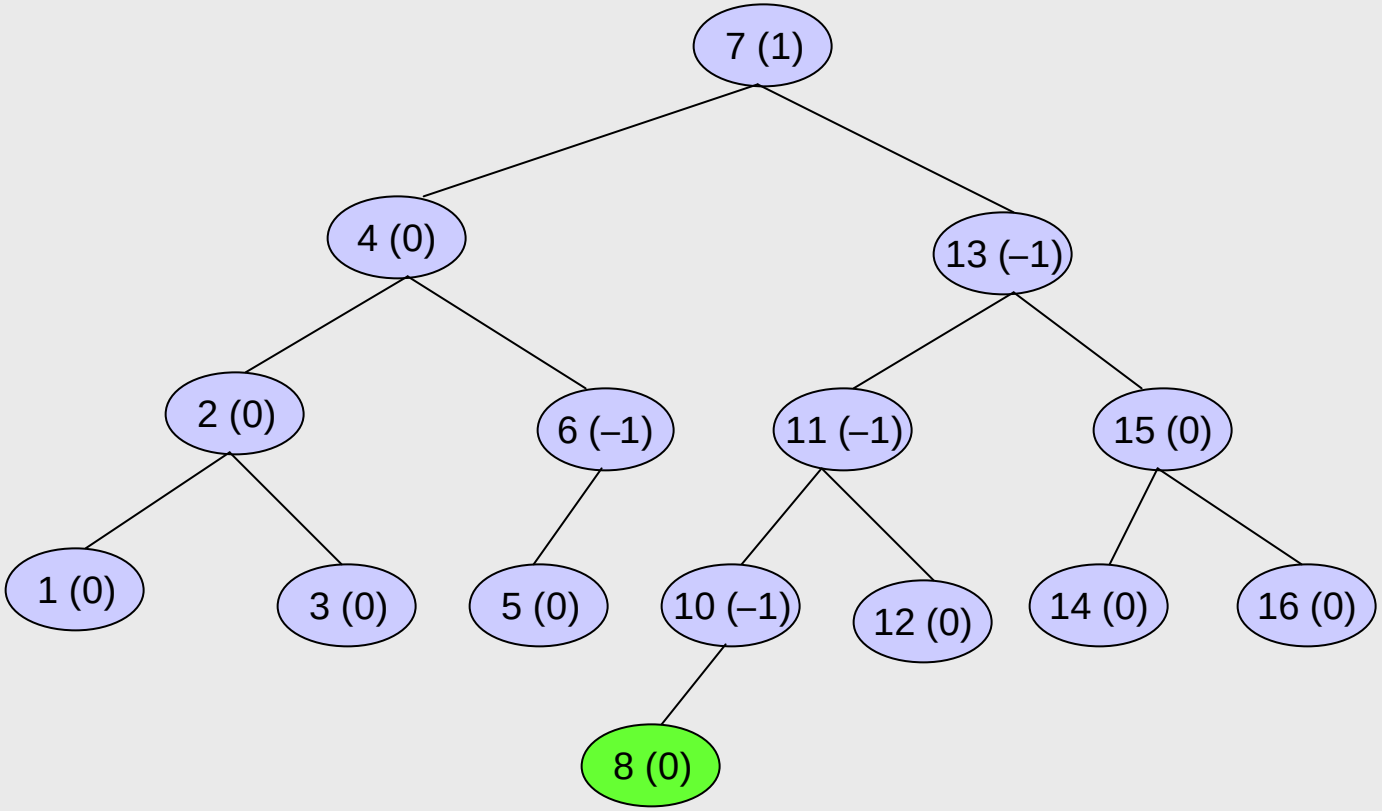
Tree has become too large to show both the insertion and rotation on the same page, so the rotation to remove the imbalance caused by the insertion of node with value 10 is shown on the next page..

Big Example – Single Right Rotation to Handle Insert of 10



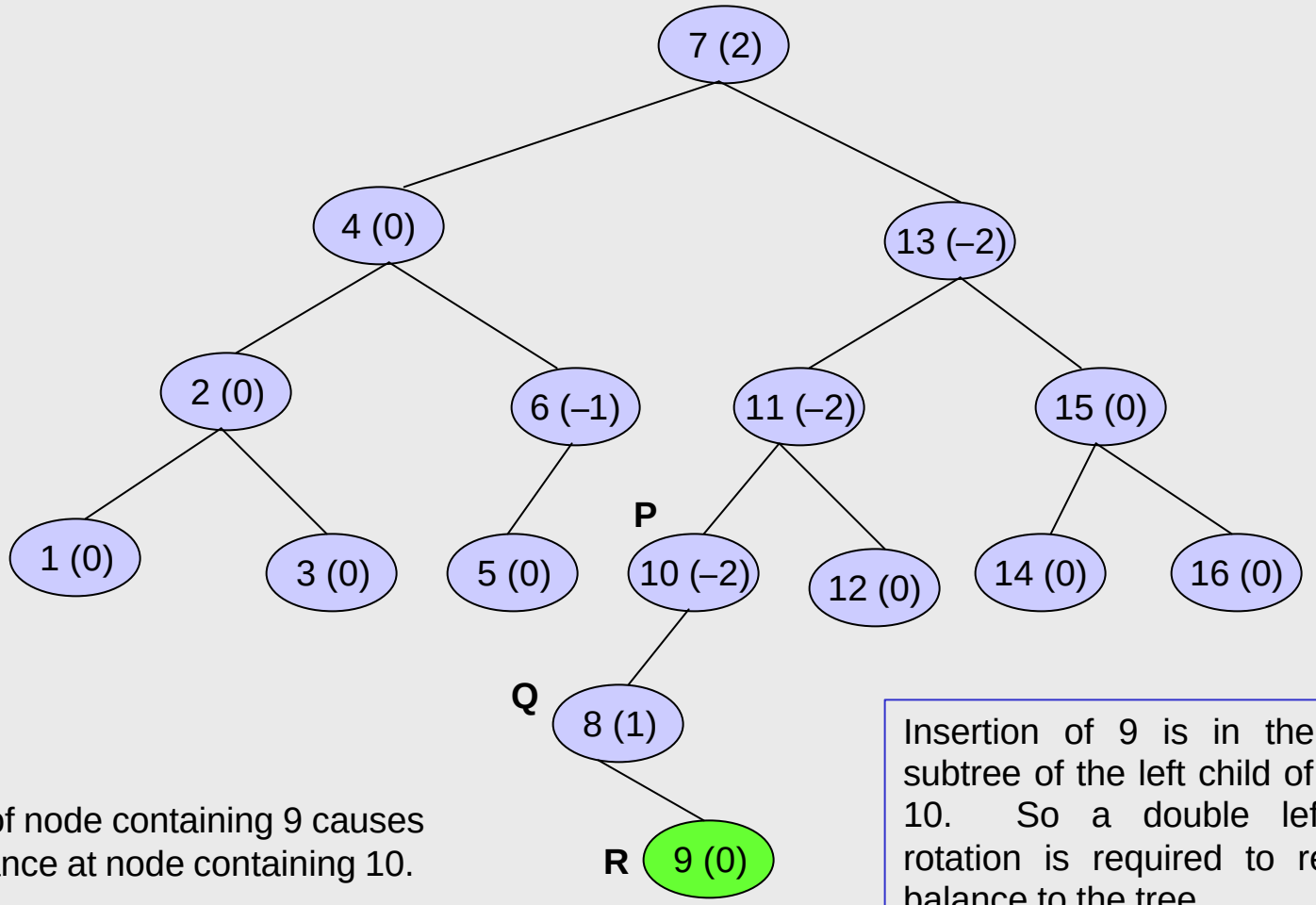
After single right rotation of 11 about 12 (Q about P).

Big Example – Insert 8



Insertion of node containing 8 causes no imbalance.

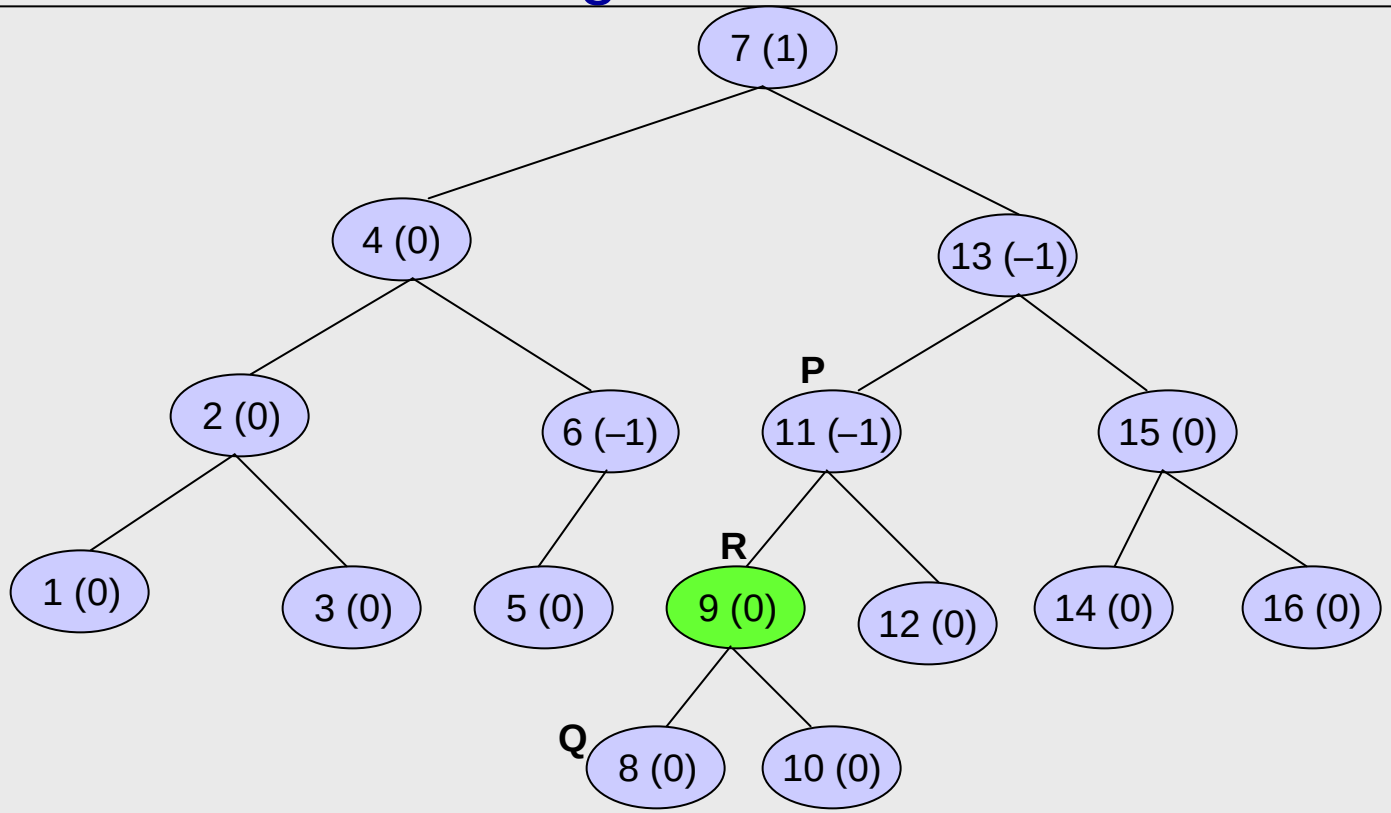
Big Example – Insert 9



Insertion of node containing 9 causes an imbalance at node containing 10.

Insertion of 9 is in the right subtree of the left child of node 10. So a double left-right rotation is required to restore balance to the tree.

Big Example – Final Tree After Double Rotation Following Insert of 9



First left rotation of 9 about 8, followed by right rotation of 9 about 10 restores balance to the tree.