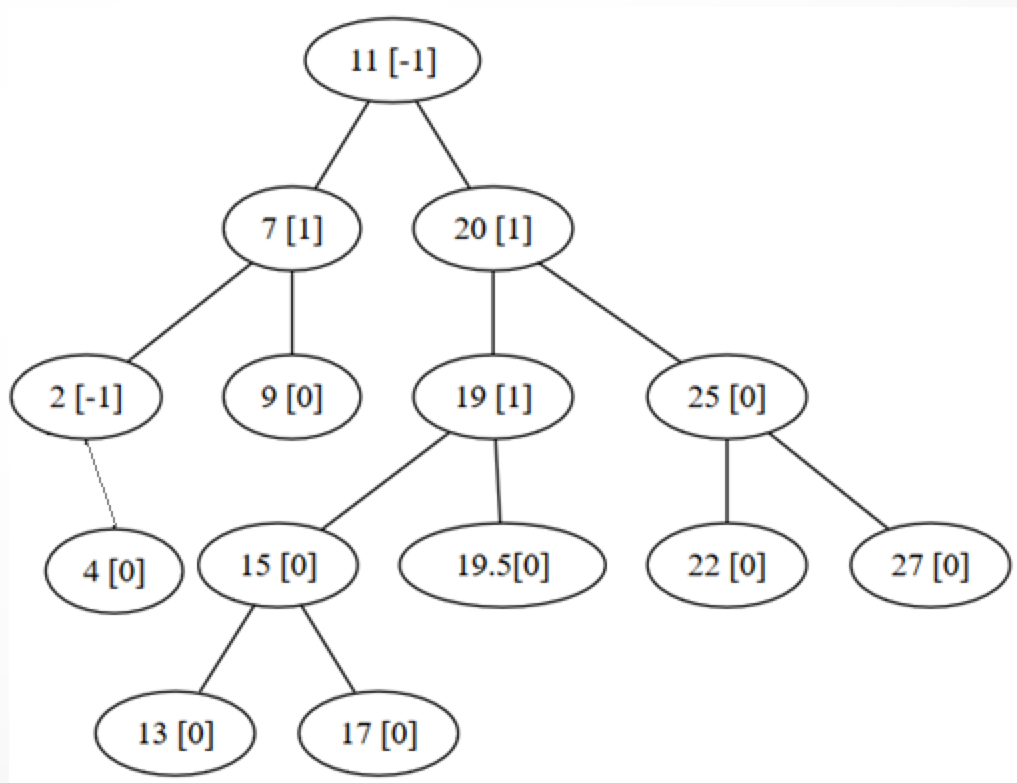
Remember: AVL trees

An AVL (Adelson-Velskii Landis) tree is a binary search tree which satisfies the following property (AVL tree property):

- If x is a node of the AVL tree:

  the difference between the height of the left and right subtree of x

  is 0, 1 or -1



Remarks:

- Height of an empty tree is -1
- Height of a single node is 0

Values in square brackets show the balancing information of a node.

Remember: AVL Tree - representation

```
AVLNode:
        info: TComp
        left: ↑ AVLNode
        right: ↑ AVLNode
        h: Integer
```

```
AVLTree:
        root: ↑ AVLNode
```

Operations:

- Search
  - Search in BST
- Insert
- Delete

Remember: AVL Trees : insert/remove

- Adding or removing a node
  - add/remove them as for an BST

  might result in a binary tree that violates the AVL tree property.

  In such cases, the property has to be restored

  ➢ Use rotations: they keep the BST property.

Properties:

- Only the nodes on the path to the modified node can change their height.
- We check the balancing information on the path from the modified node to the root. When we find a node that does not respect the AVL tree property, we perform a suitable rotation to rebalance the (sub)tree.

Remember: AVL - insert

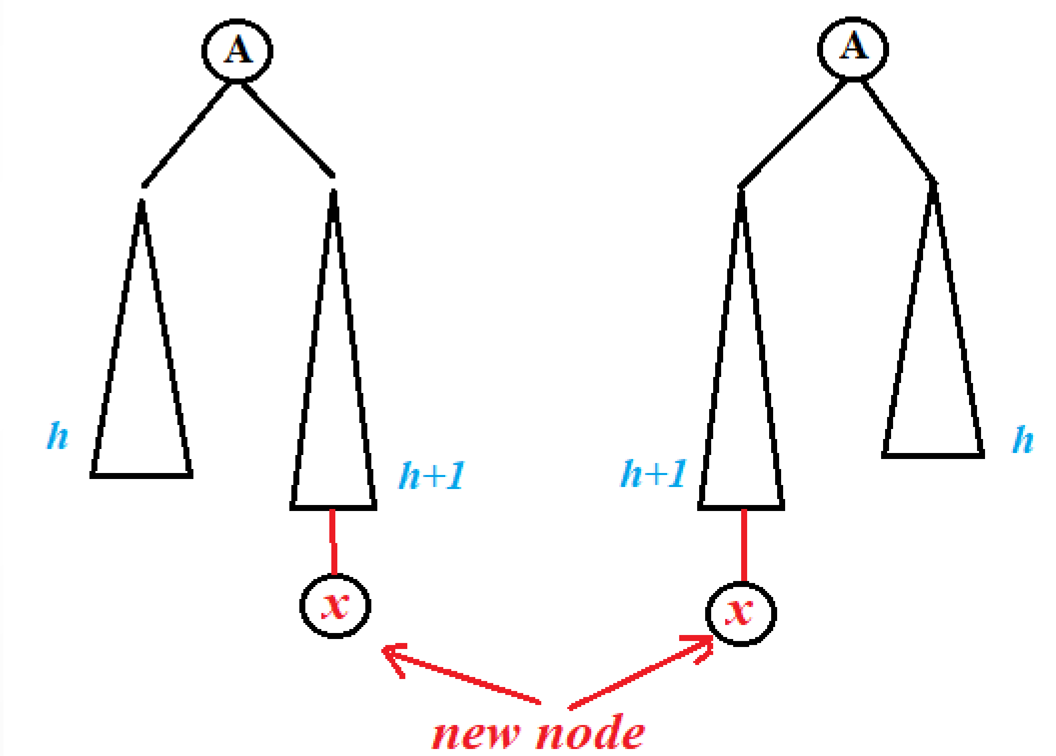
Insertion:

- insert an element like in BST case
- rebalance the tree (if it is the case)

  consider all the ancestors (to the root)

  rebalance → one or more tree rotations.

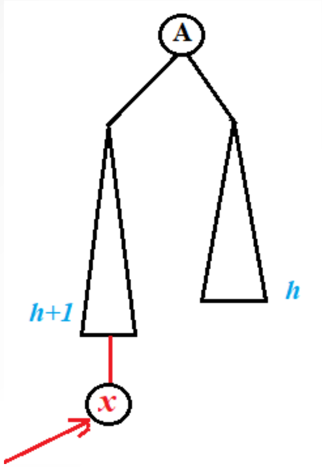
When to rebalance:



AVL Trees – insert cases

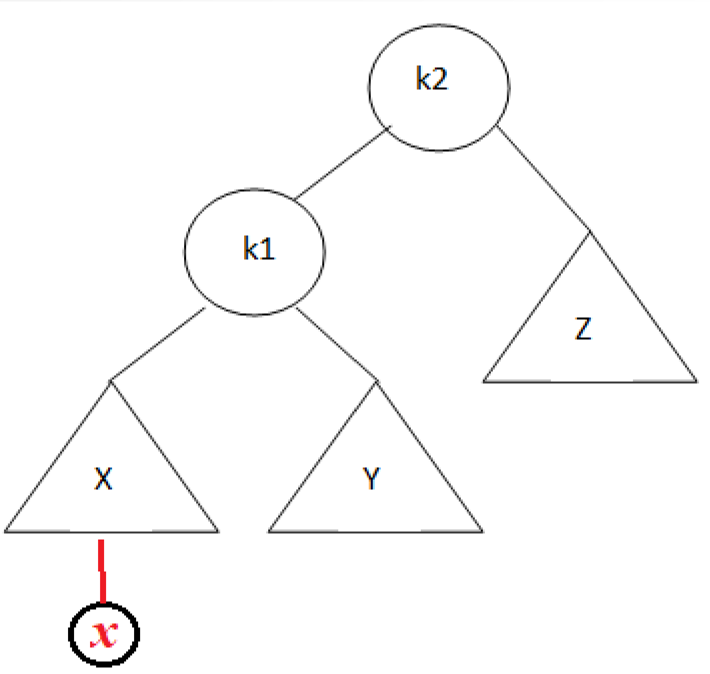
Case 1:

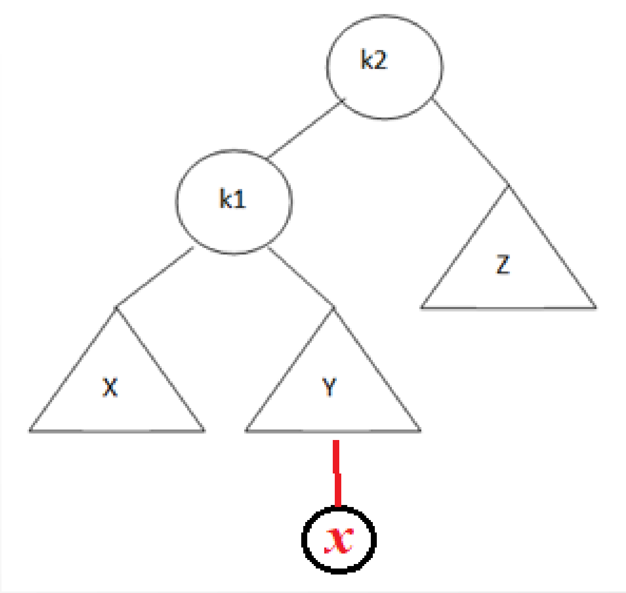
- RotateRight(k2)



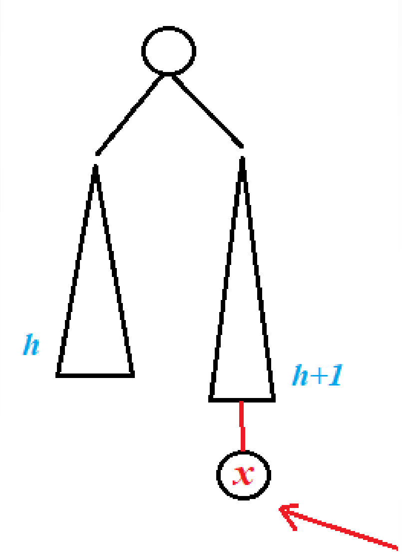
Case 2:

- DoubleRotateRight(k2)



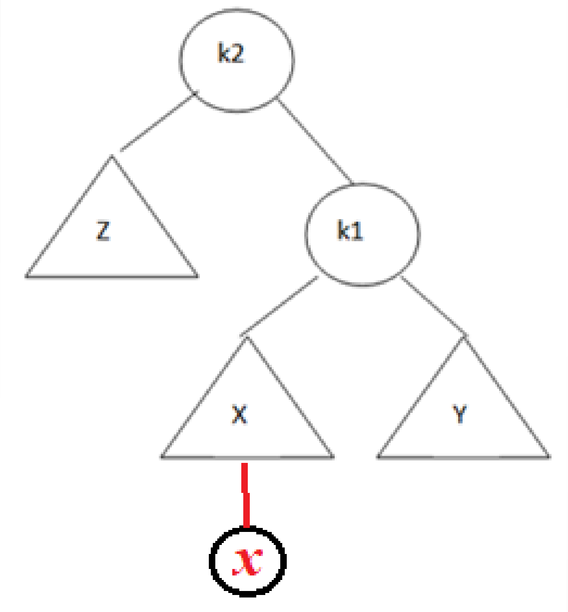


AVL Trees – insert cases



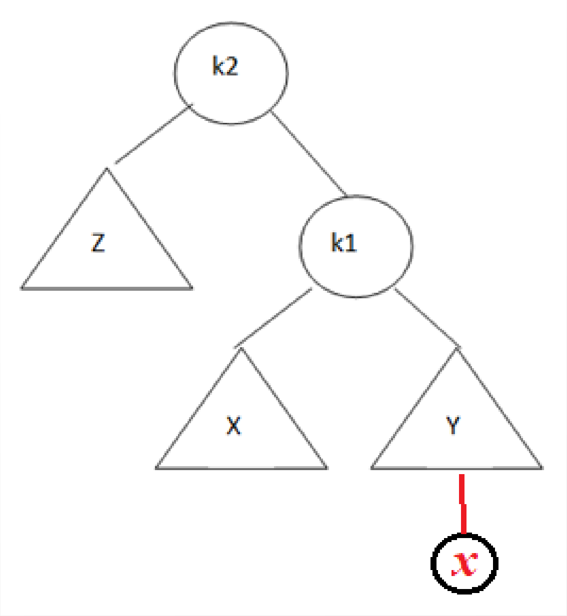
Case 3:

- DoubleRotateLeft(k2)



Case 4:

- RotateLeft (k2)



Pre: el - element, p - AvlTreeNode
Post: return the new p

```
Function insert_rec(p , el)
    if( p = NIL)
        p ← createNode(el)
    else
        if( el <= [p].info) then
            [p].left ← insert_rec([p].left , el )
            if (Height( [p]. left ) - Height([p]. right) = 2 )
                if( el <= [[p].left].info)
                    p ← RotateRight ( p )
                else
                    p ← DoubleRotateRight ( p )
                endif
            endif
        else   // el > [p].info
            [p].right ← insert_rec([p].right , el )
            if( Height( [p].right ) - Height( [p].left ) = 2 )
                if( el > [[p].right].info ) then
                    p ← RotateLeft ( p )
                else
                    p ← DobleRotateLeft( p );
                endif
            endif
        endif
        [p].h ← Max( Height( [p].left ), Height( [p].right ) ) + 1;
    endif
    insert_rec ← p
End_function
```

```
Function createNode ( el)
    allocate(p)
    [p].info ← el
    [p].h ← 0;
    [p].left ← NIL
    [p].right ← NIL
    createNode ← p
end_function
```

```
Subalg. insert(T , el)
        p ← T.root
        T.root ← insert_rec(p, el)
end_subalg.
```

Delete(k)

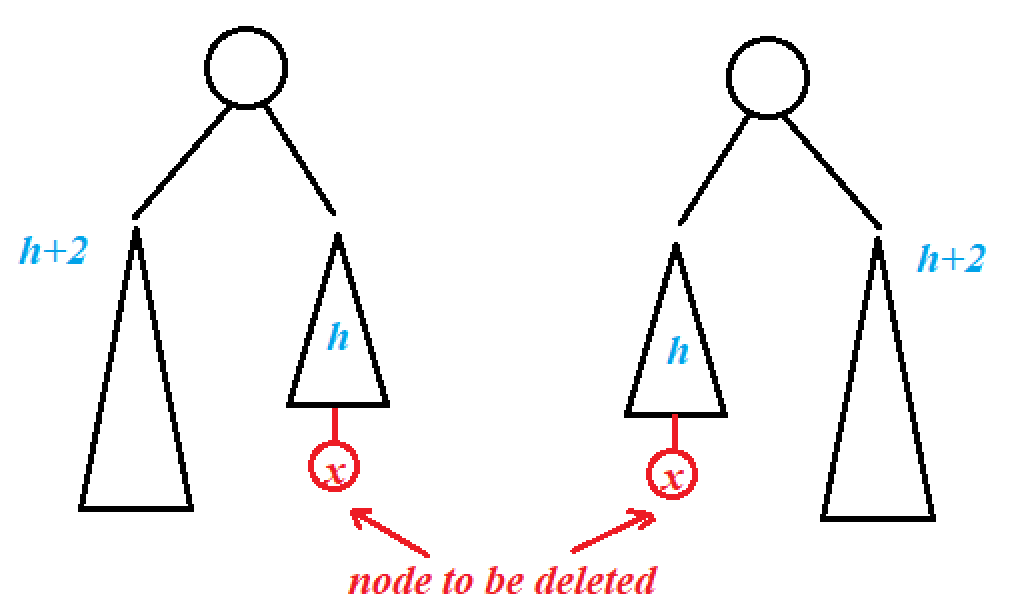
- find the node x where k is stored
- delete the contents of node x ~ similar with BST

  Deleting a node in an AVL tree can be reduced to deleting a leaf

- rebalance:

  go from the deleted leaf towards the root

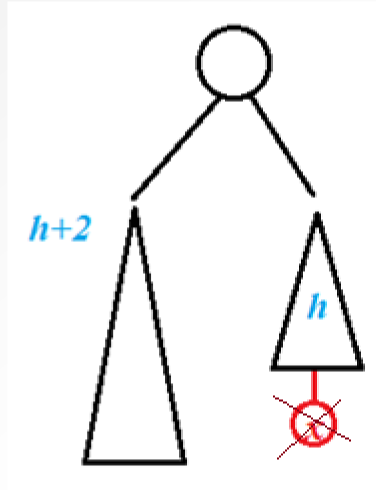
  and rebalance with rotations if necessary.



AVL Trees – remove cases

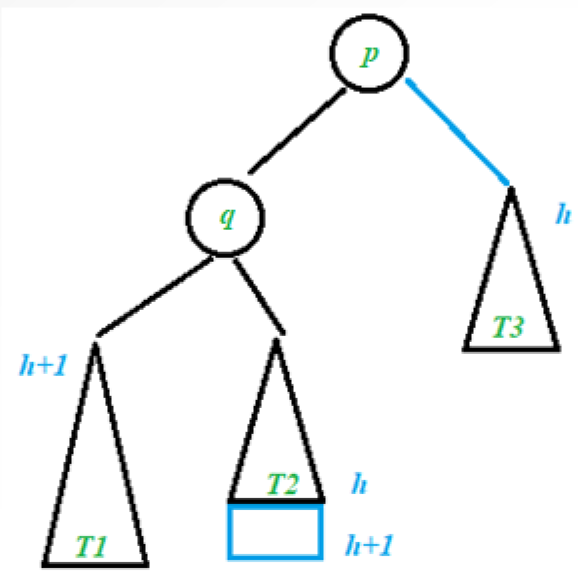
Case 1:

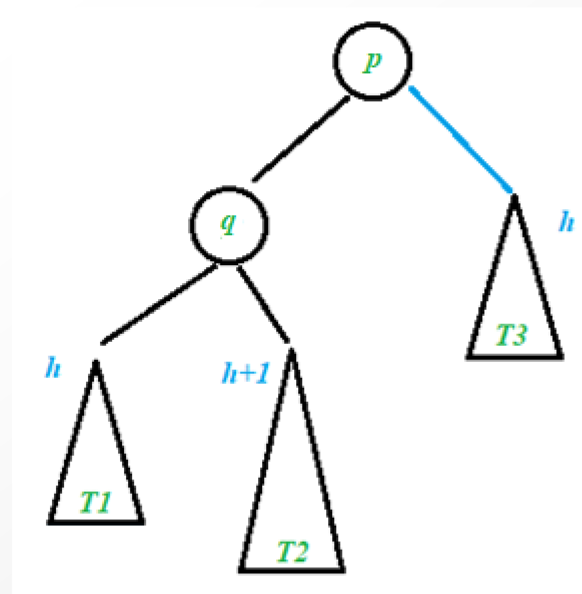
- RotateRight(***p***)



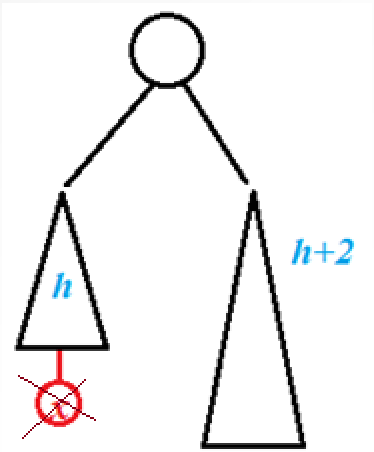
Case 2:

- DoubleRotateRight(***p***)



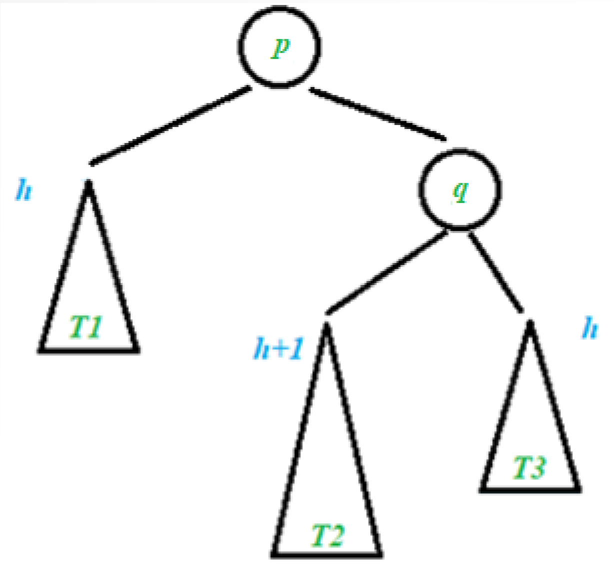


AVL Trees – remove cases



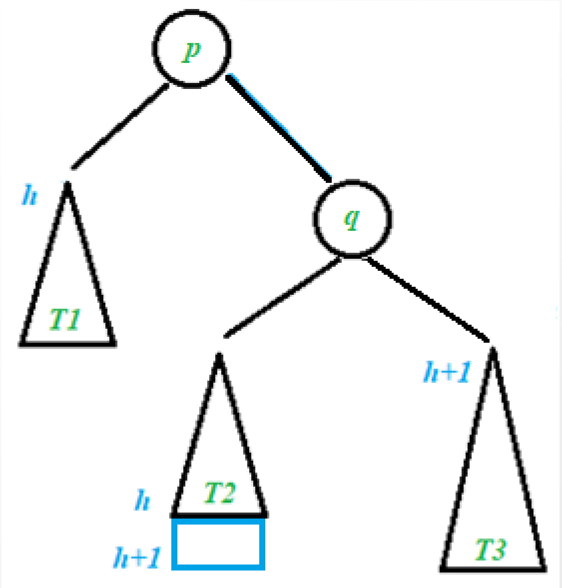
Case 3:

- DoubleRotateLeft(***p***)



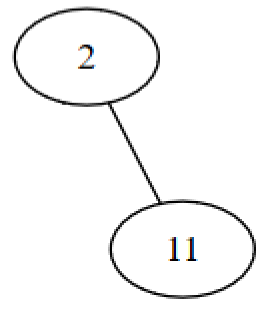
Case 4:

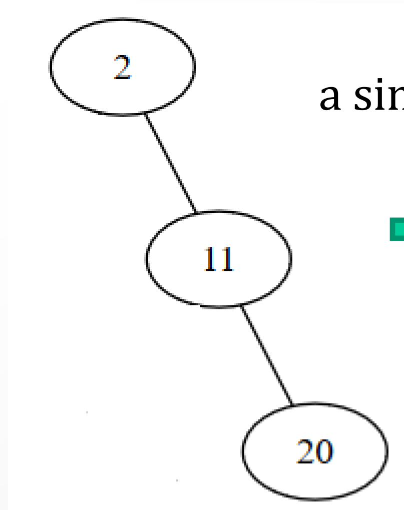
- RotateLeft(***p***)



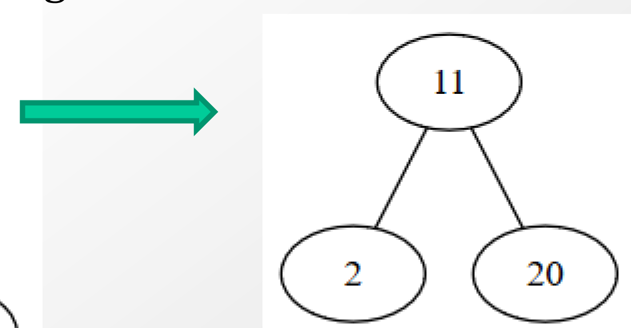
AVL insert: example

- Start with an empty AVL tree
- Insert 2
- Insert 11
- Insert 20
- Insert 7 ...



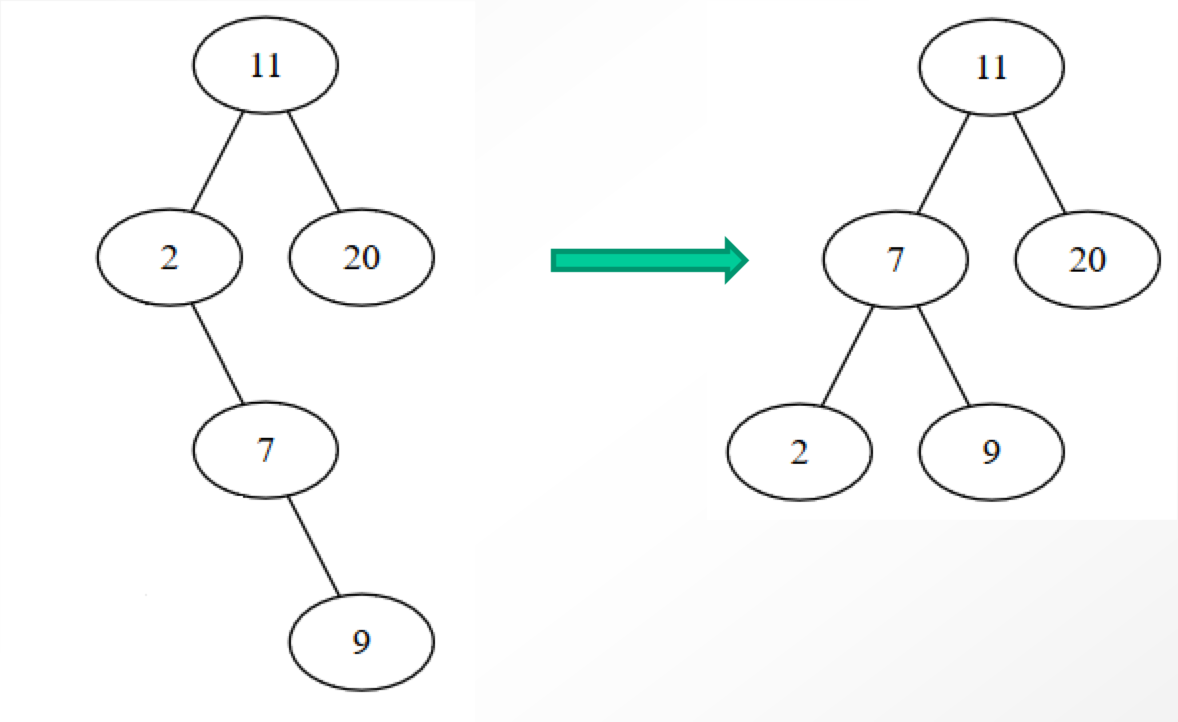


a single left rotation on node 2



AVL insert: example

- Operation:
  Insert 9



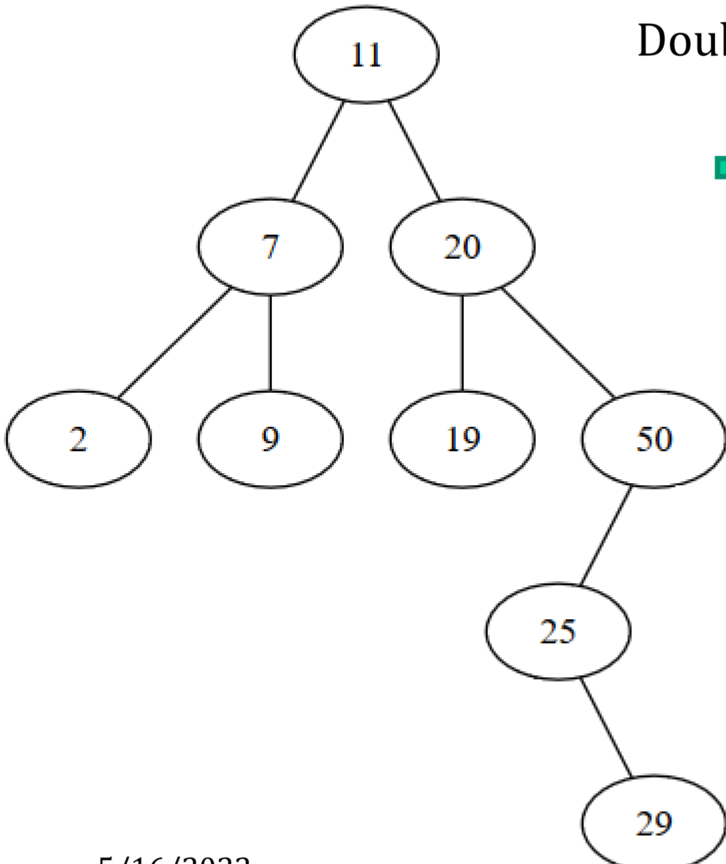
a single left rotation on node 2

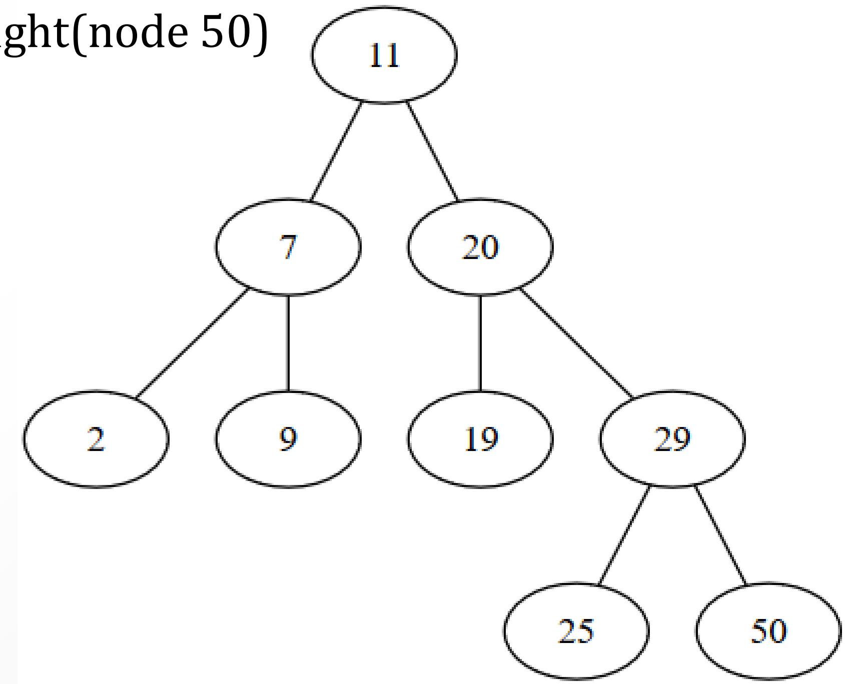
- Insert 50
- Insert 19
- Insert 25
- Insert 29

AVL insert: example

- Operation: insert 29

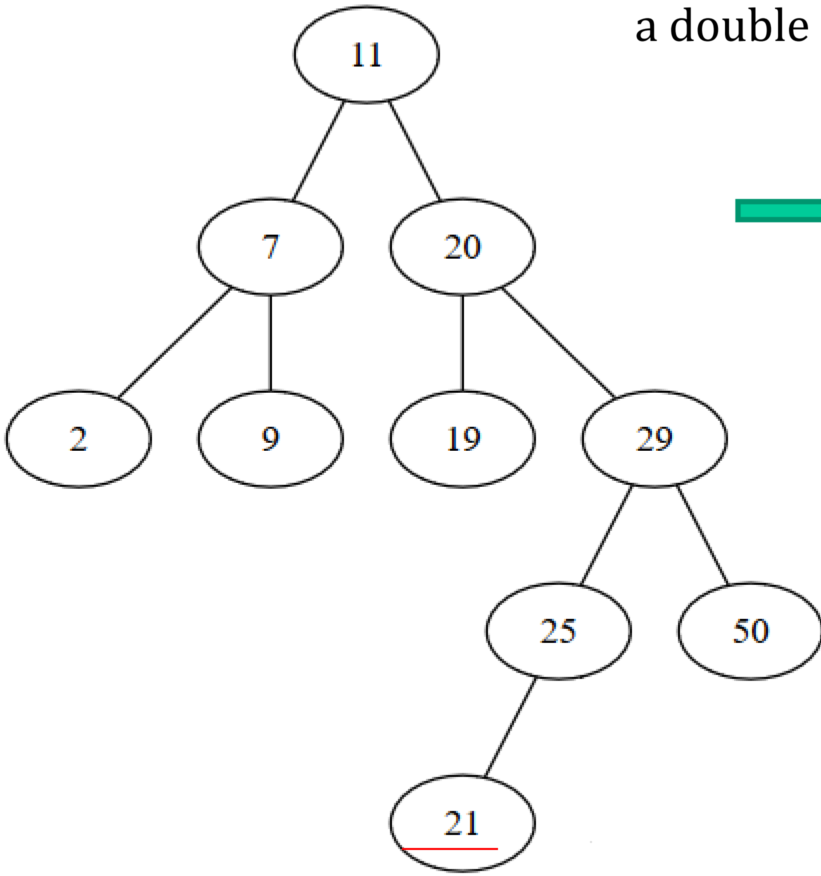
a double right rotation on node 50
DoubleRotateRight(node 50)



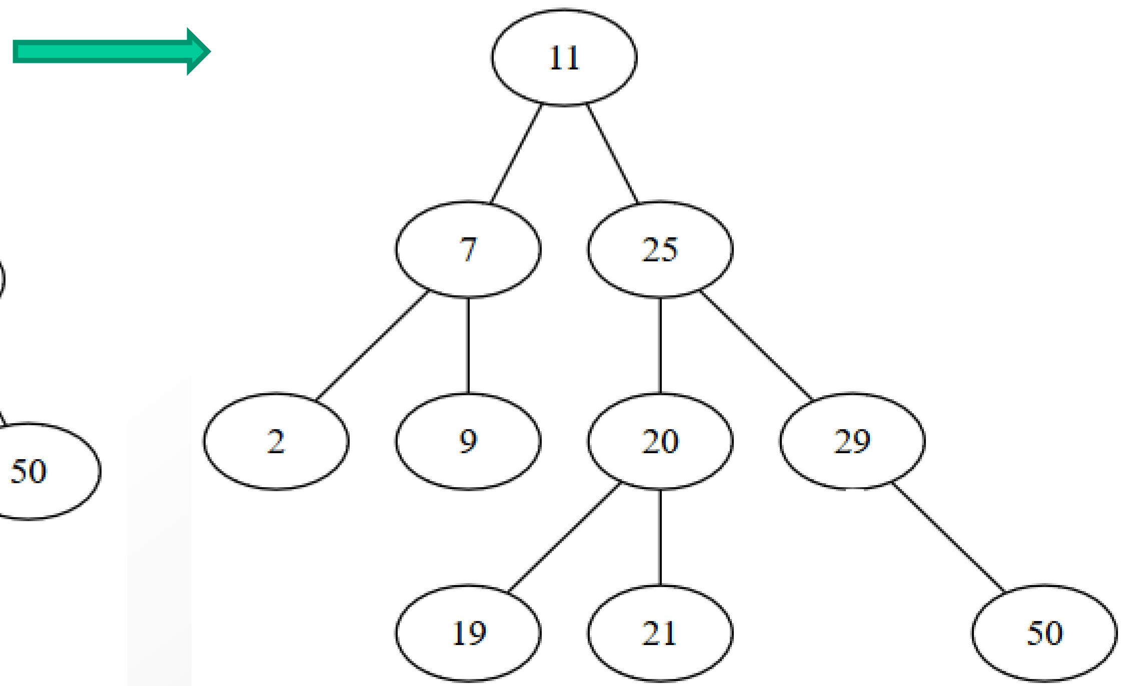


AVL insert: example

- Operation: add 21 to the previous tree

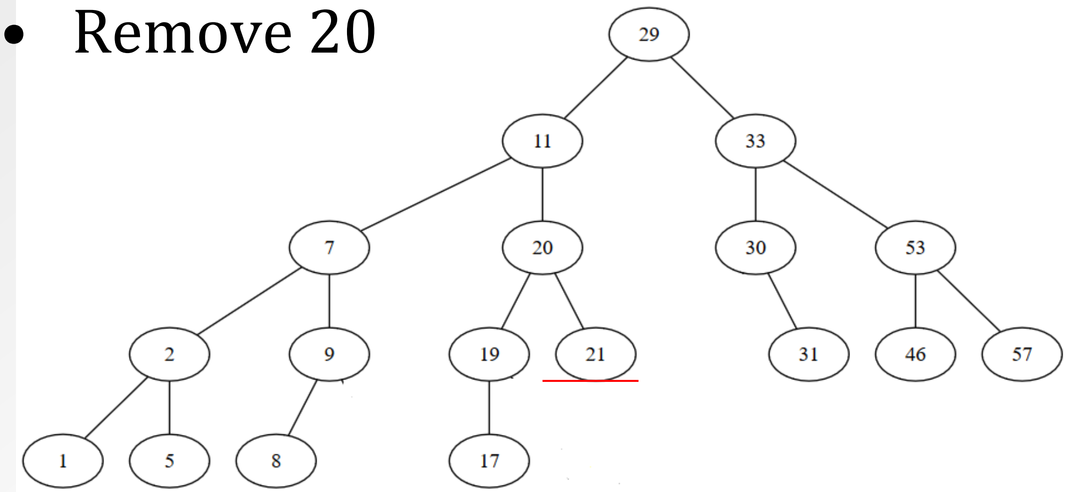


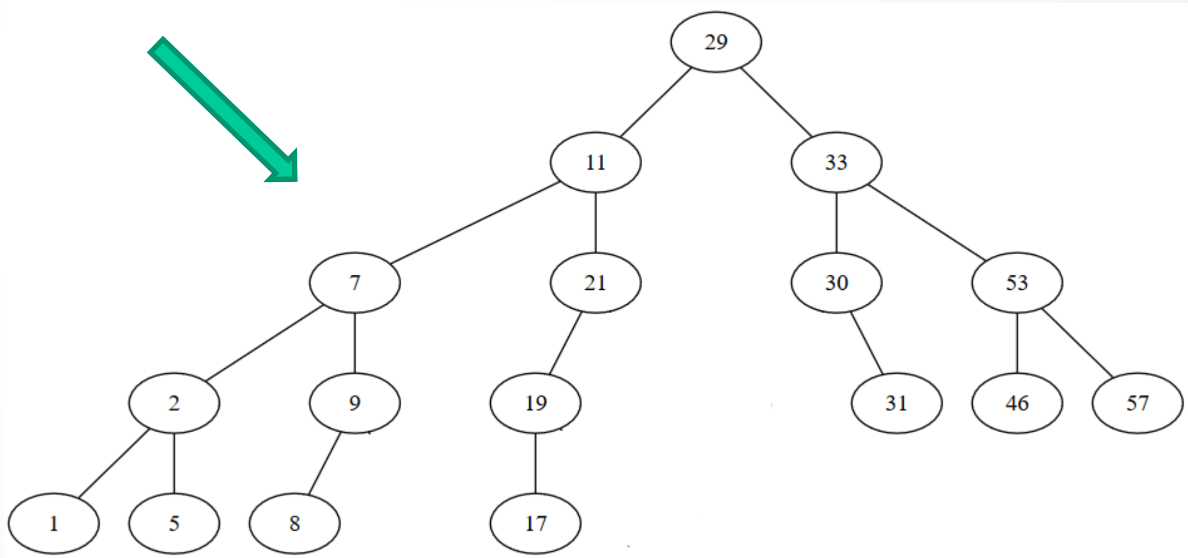
a double left rotation on node 20



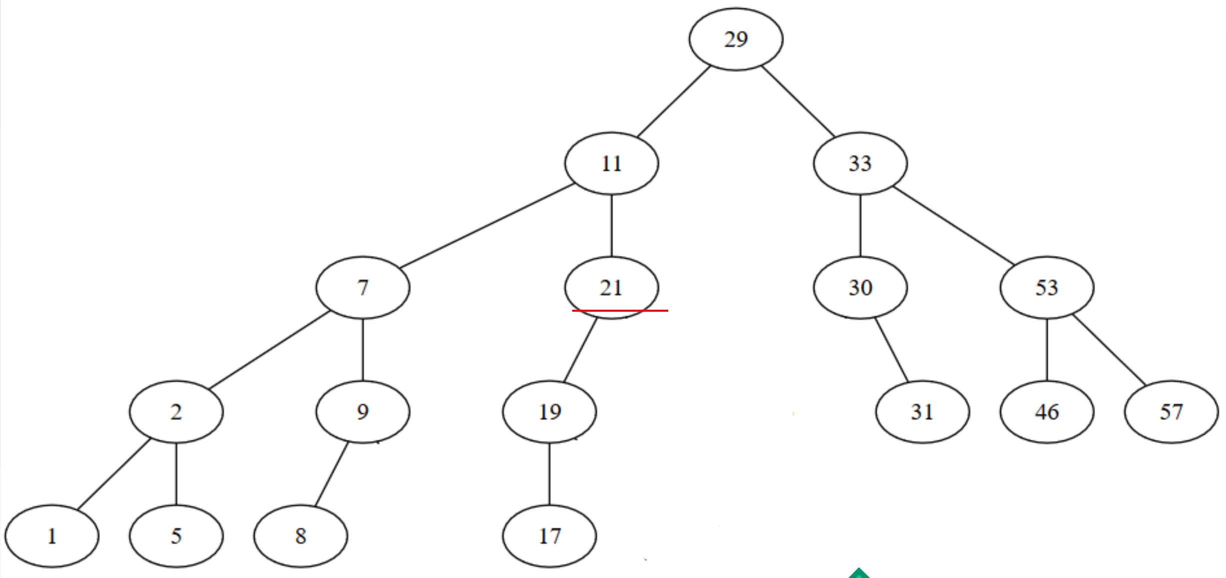
AVL - example of remove

- Remove 20

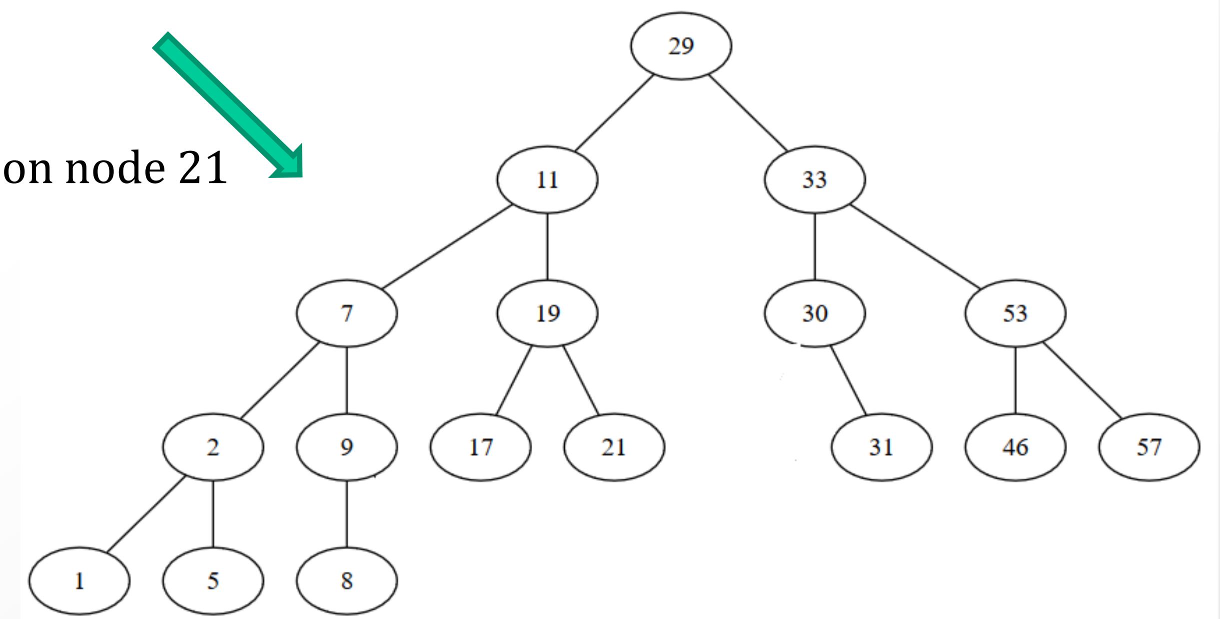




AVL - example of remove

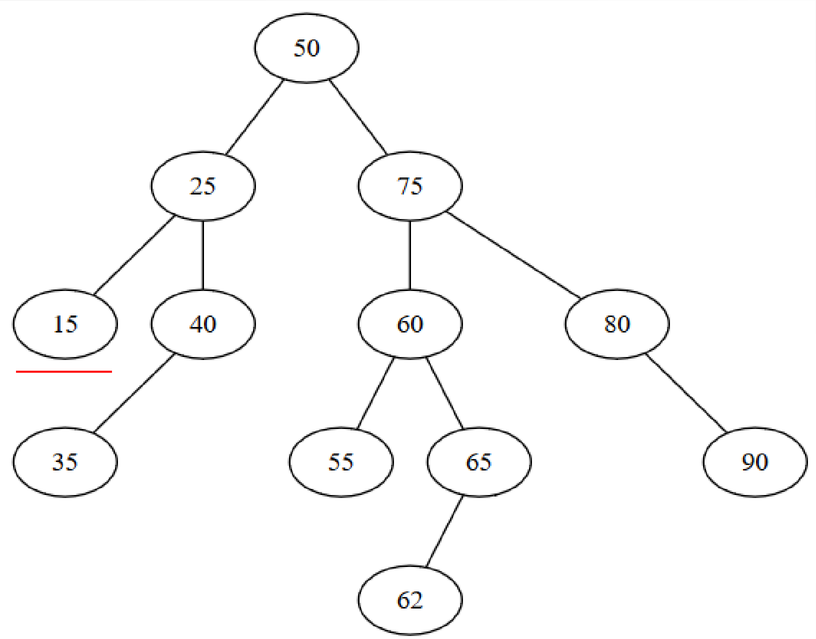


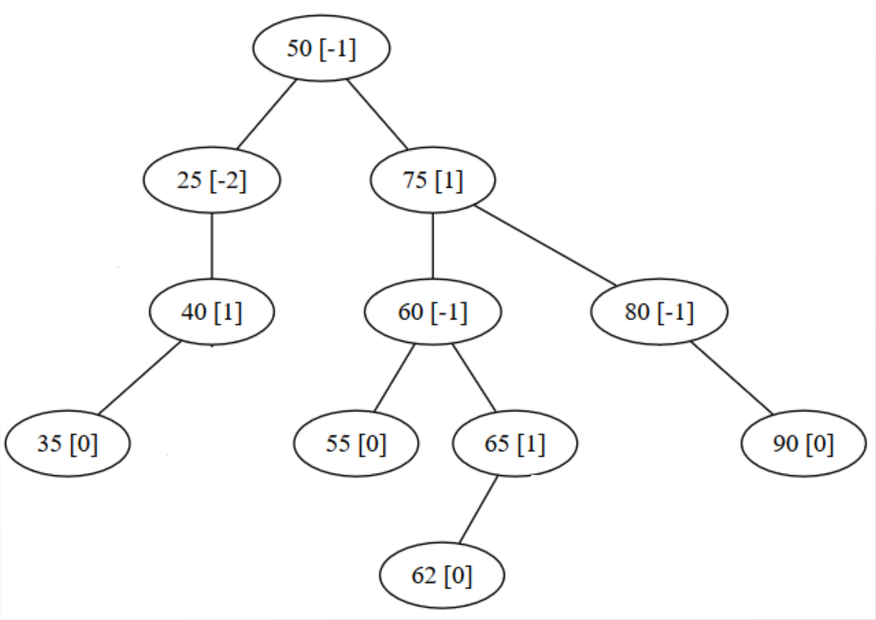
a single right rotation on node 21



AVL - example of remove

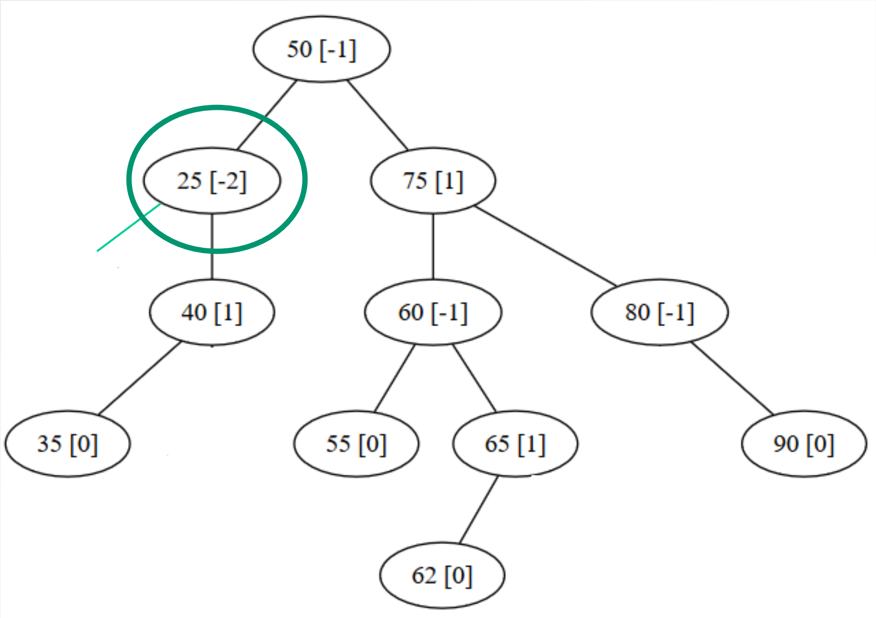
- Remove 15

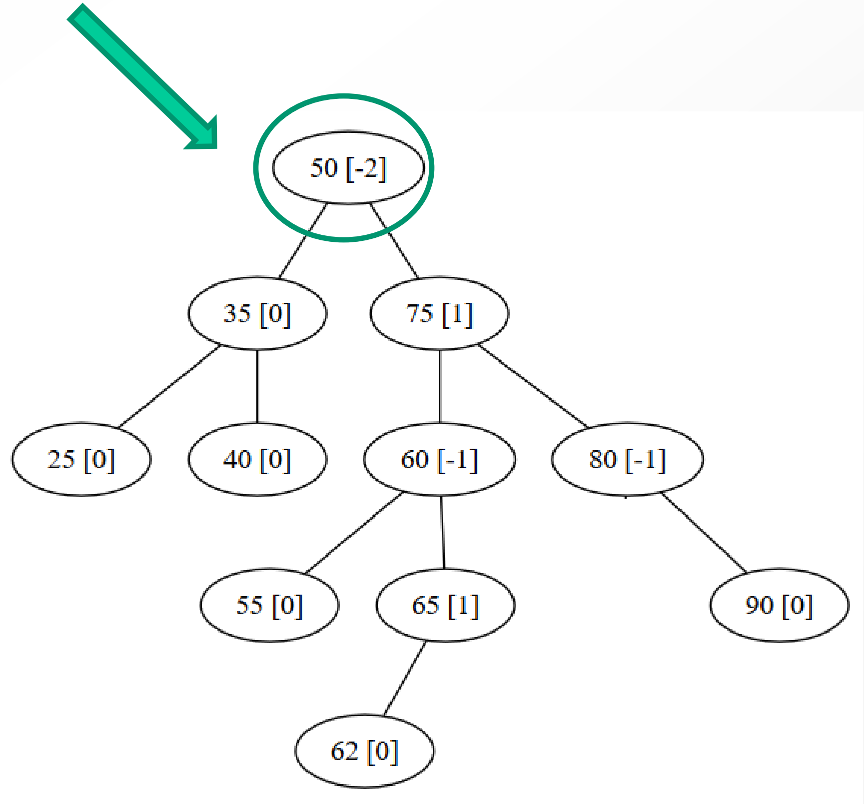




• .

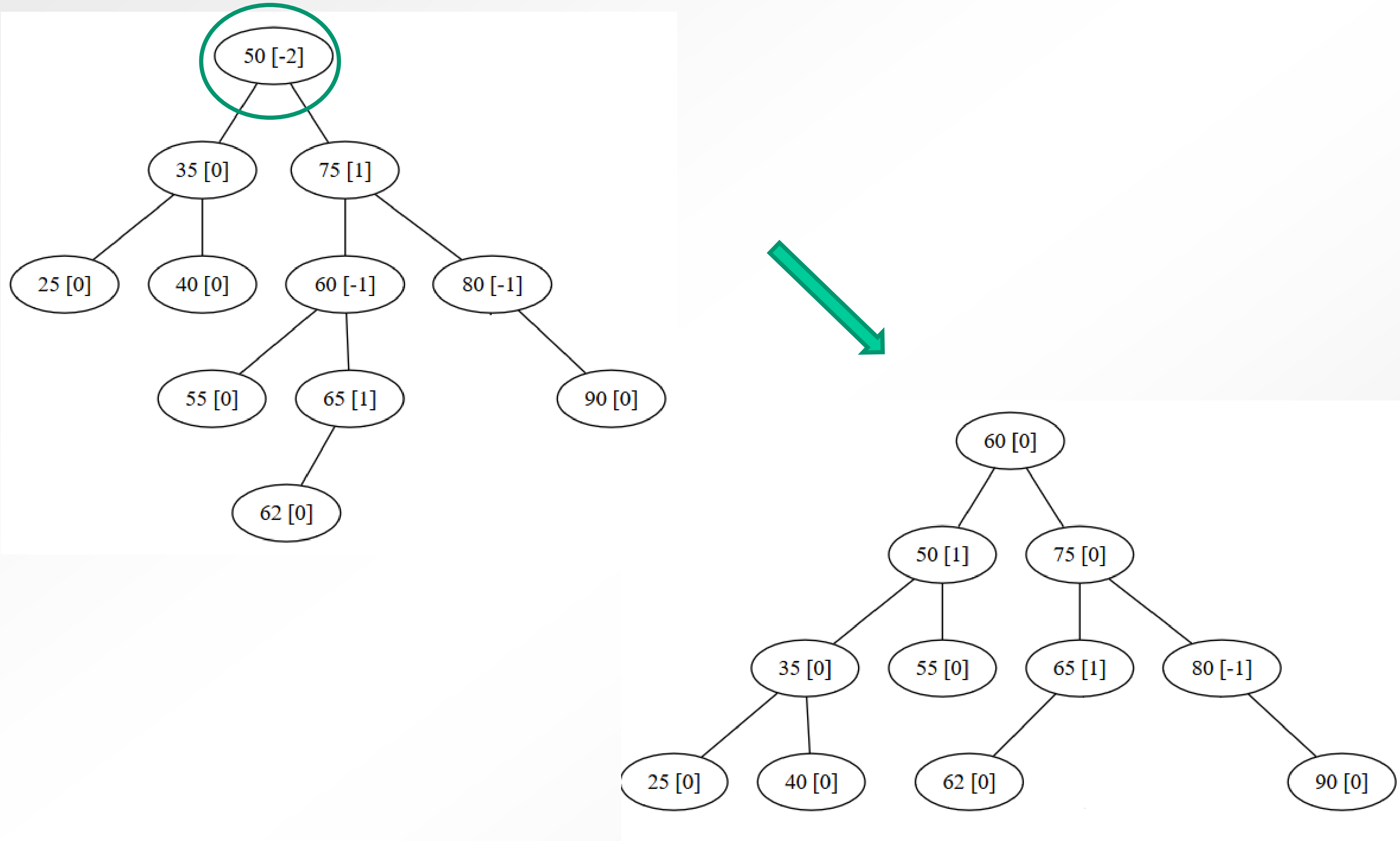
AVL - example of remove



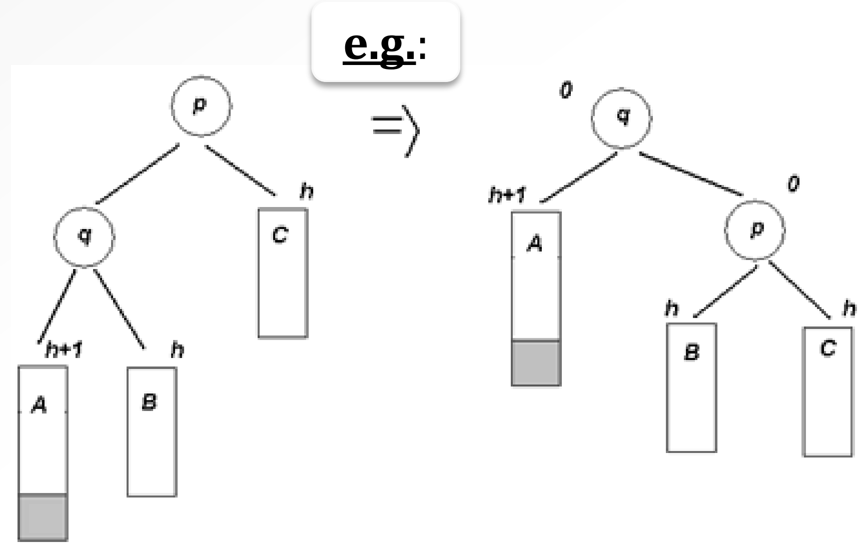


- .

AVL - example of remove



RotateRight (AVL)



Function RotateRight (p)

q ← [p].left

[p].left ← [q].right

[q].right ← p

[p].h ← Max(Height([p].left), Height([p].right)) + 1

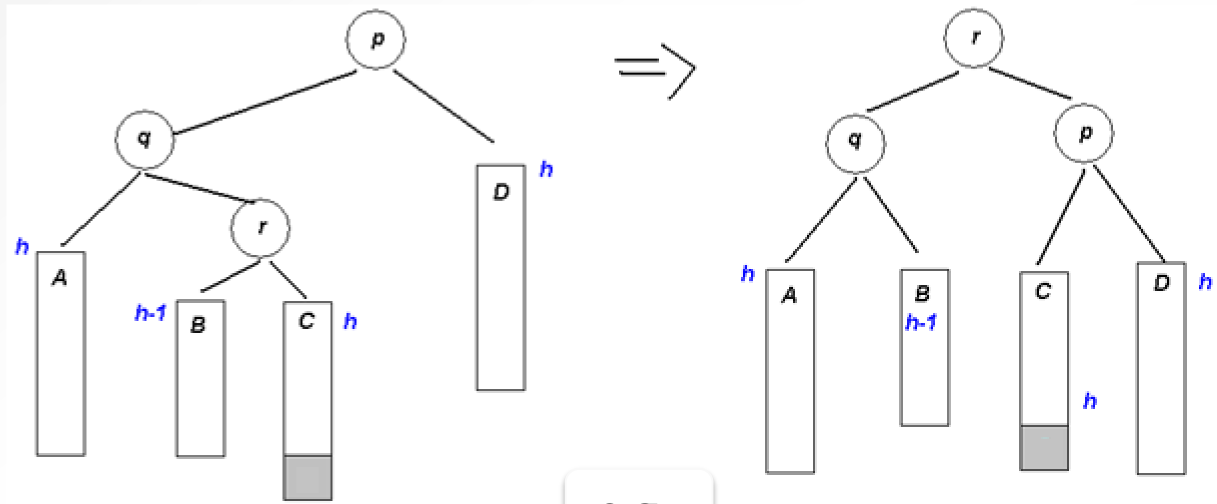
[q].h ← Max(Height([q].left), Height ([q].right)) + 1

RotateRight ← q

end_RotateRight

Simlar algorithm for: function RotateLeft (p)

DoubleRotateRight (AVL)



e.g.:

Function DoubleRotateRight (p)

q ← [p].left

[p].left ← RotateLeft (q)

DoubleRotateRight ← RotateRight (p)

end_function

Other names for DoubleRotateRight :

- Double right rotate around p
- Double right rotation around p
- Double left-right rotation around p

Similar for: DoubleRotateLeft(p)