AVL TREES

Notes by Yan Yan

Outline

- Review – Binary Search Tree
- Why AVL Trees
- Definition of AVL Trees
- Rotation and Restore AVL Trees

Learning Objectives

- Define AVL trees
- Determine if a tree is an AVL tree
- Restore AVL trees with tree rotations

Binary Search Tree (complexity)

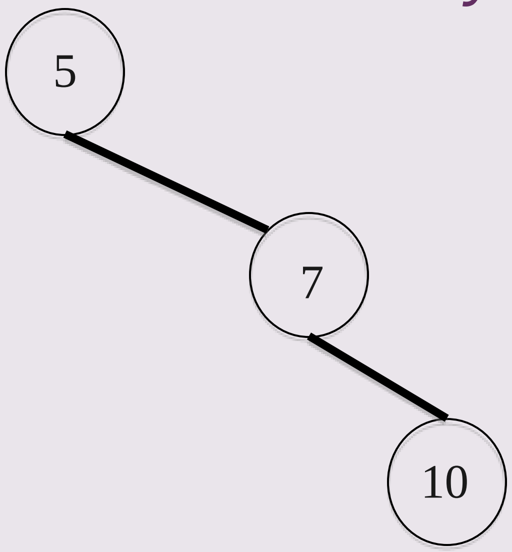
- Binary search tree
  - *Search, insert, and delete*

- Question: What are the time complexities of these operations?

Binary Search Tree (complexity)

- Binary search tree
  - *Search, insert, and delete*

- Question: What are the time complexities of these operations?
  - *What the tree may look like in the worst case?*



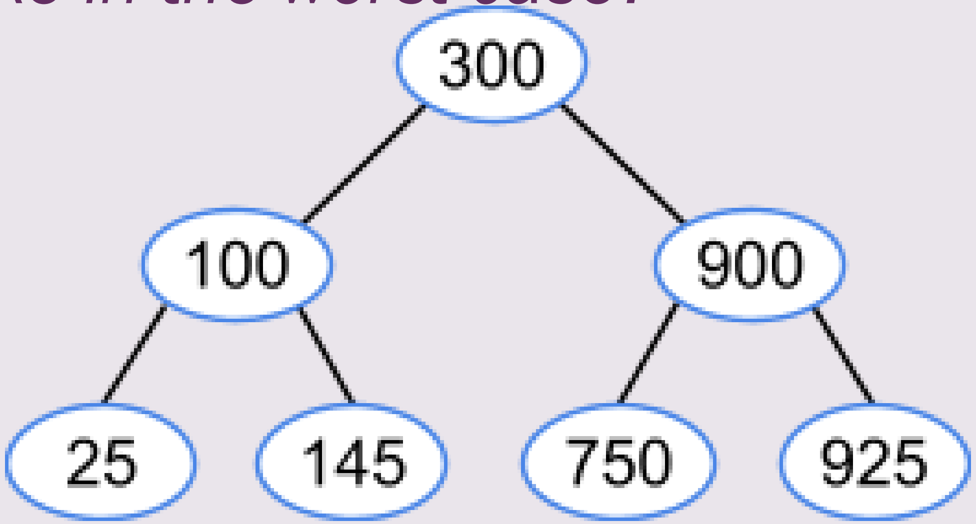
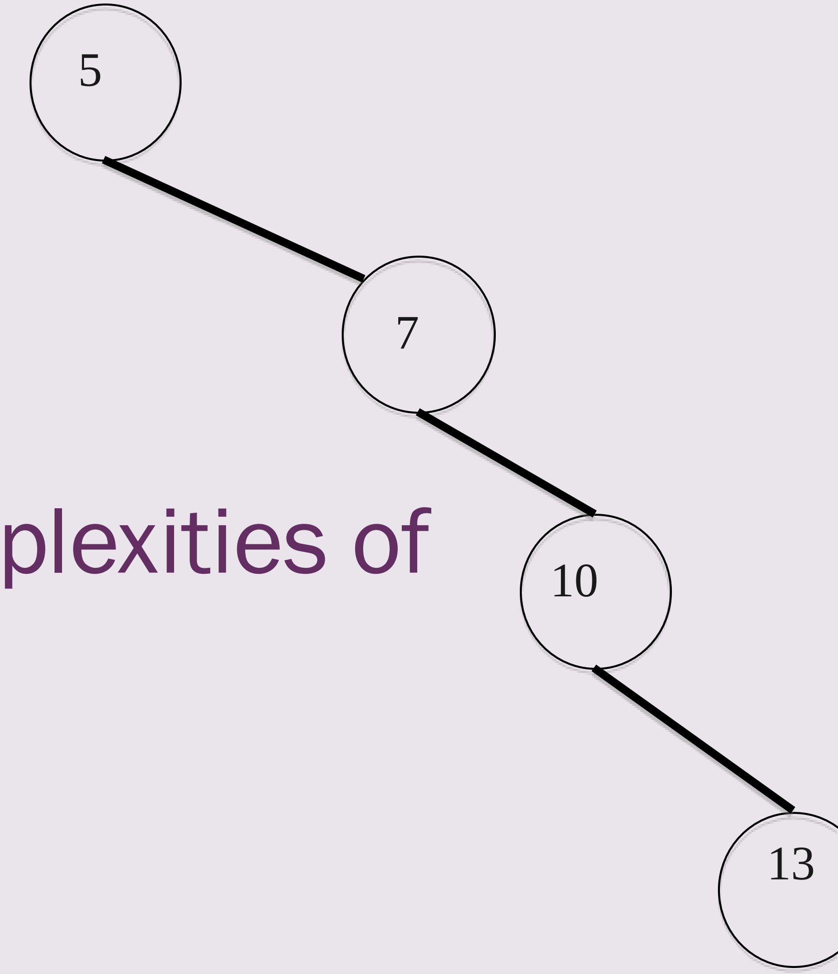


Figure from 7.3 7.3 Binary search trees

Binary Search Tree (complexity)

- Binary search tree
  - *Search, insert, and delete*

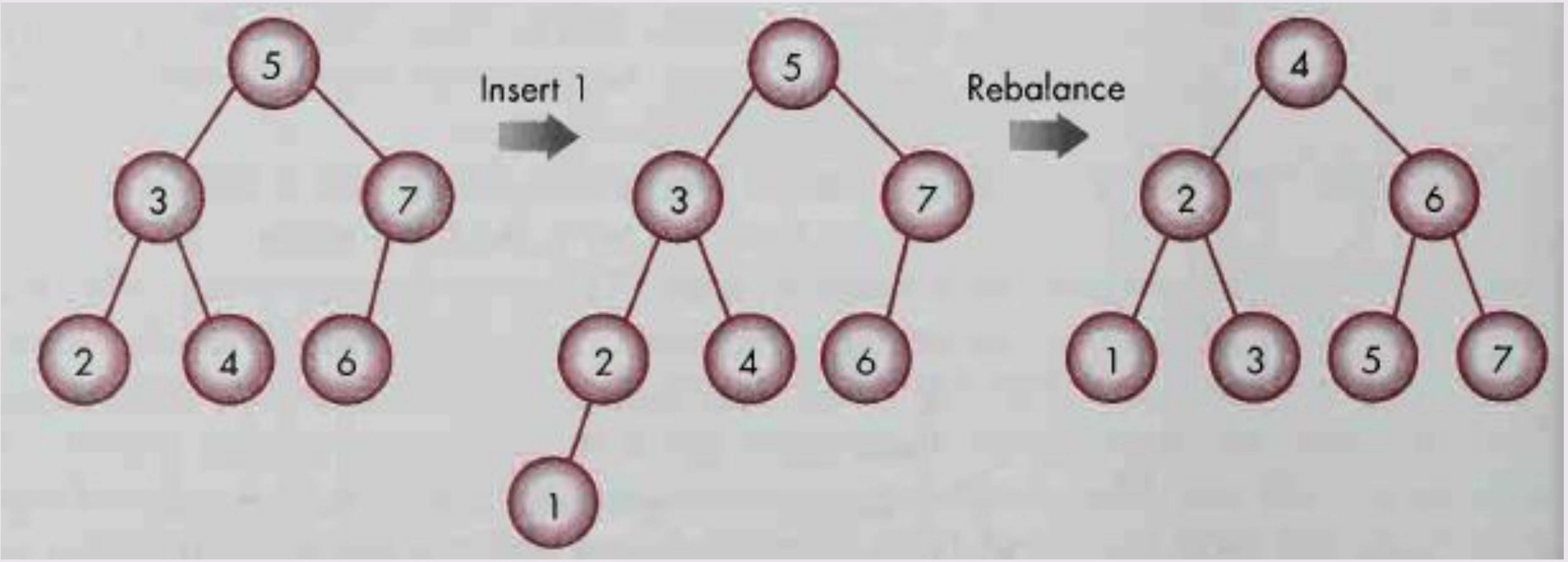
- Question: What are the time complexities of these operations?
  - *Worst case O(n)*



Why AVL Tree?

A binary tree is balanced if the height of the tree is **O(Log n)** where n is the number of nodes.

- We want to avoid the worst case in the binary search tree
- How? -- balance the tree
- Issue -- rebalance the tree may take up O(n) operations



The above figure is the Figure 9.41 from the book "Thomas A. Standish: Data Structures, Algorithms & Software Principles in C, Addison Wesley"

Why AVL Tree?

- Is there some other way to (almost) balance the tree with no more than O(logn)?

Why AVL Tree?

- Is there some other way to (almost) balance the tree with no more than O(logn)?
- YES!
- AVL Trees
  - *Named after Adelson-Velskii and Landis*

AVL Tree Definition

- Height – longest path from the root to some leaf
  - *An empty tree has height 0 (note: some other definitions make this as -1)*
  - *A tree with a single node has height 1 (note: some other definitions make this as 0)*

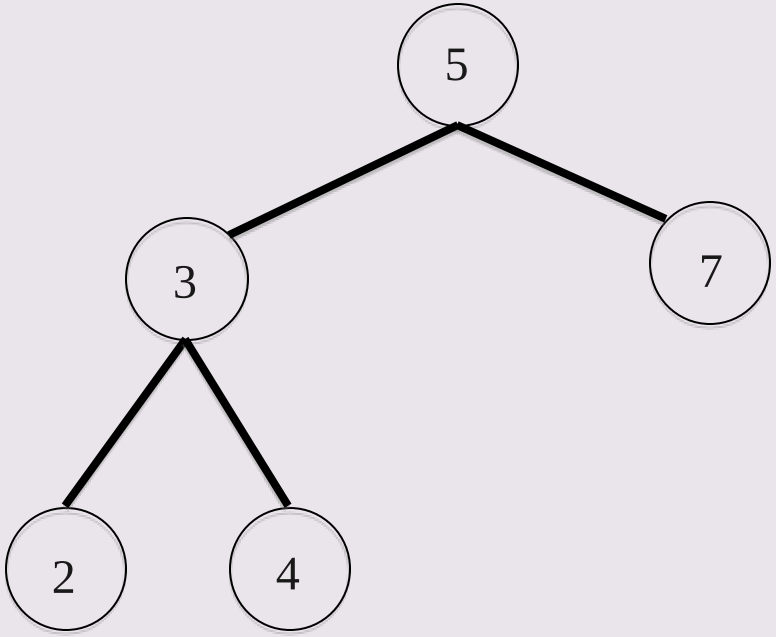
- AVL property: If N is a node in a binary tree T, we say that node N has the **AVL property** if the heights of the left and right subtrees of node N are either equal or if they differ by 1.

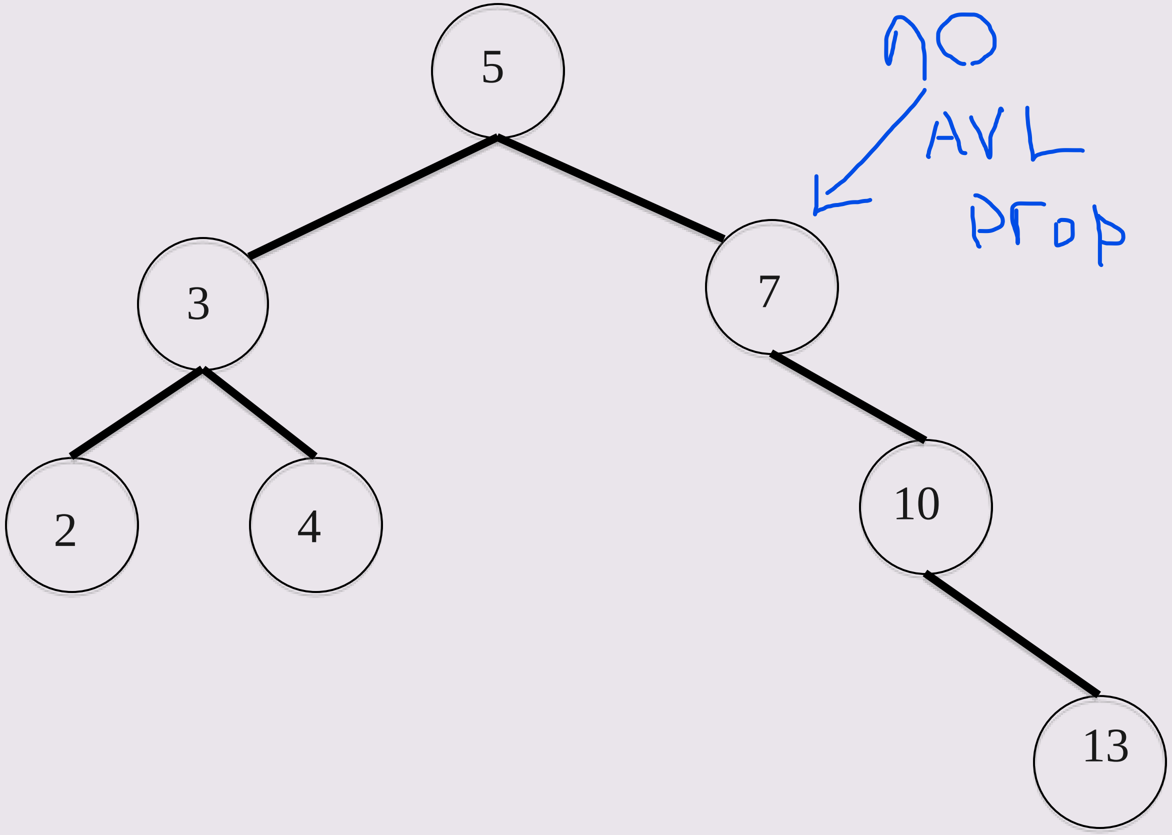
- Balance factor of node N, BF(N) = Height_(Nleft) - Height_(Nright)

- AVL Tree: A binary tree that each of its nodes has AVL property

AVL Tree Definition

- Question: are the following trees AVL Trees?

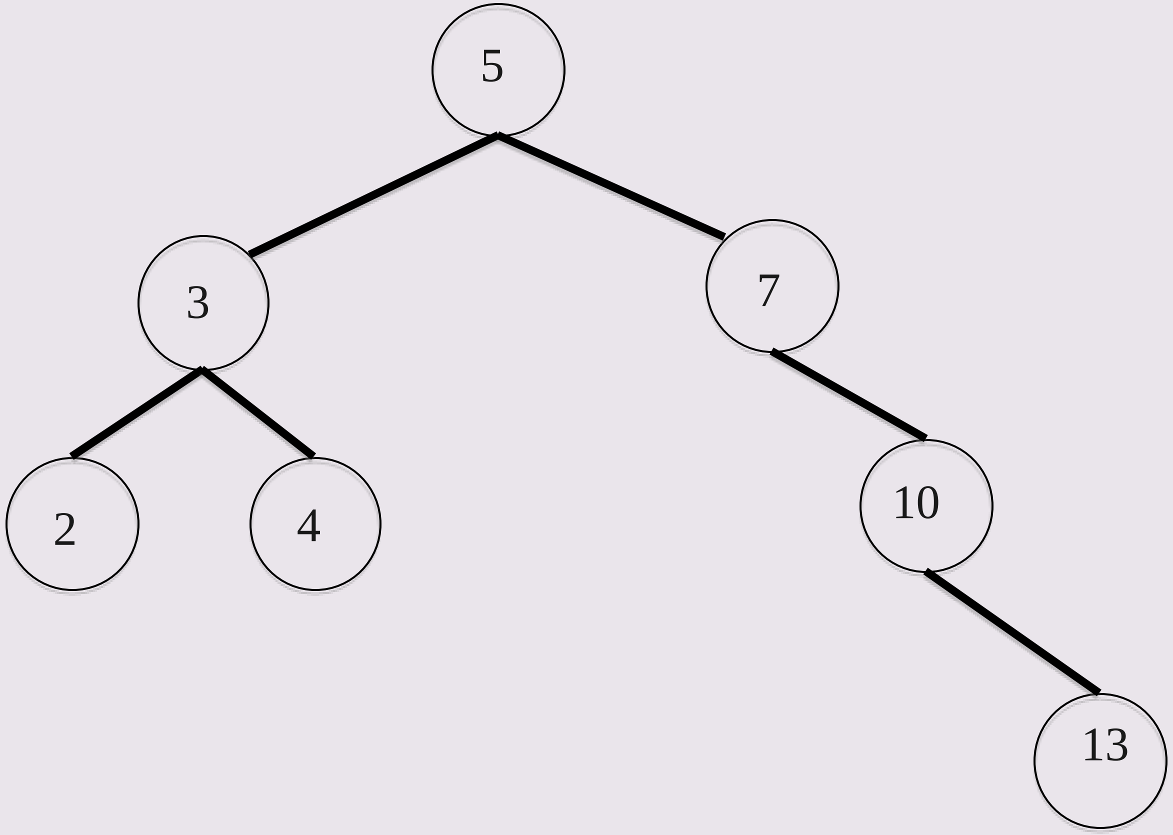




BF(N) = -1, 0, or 1

Restore AVL Property – example 1

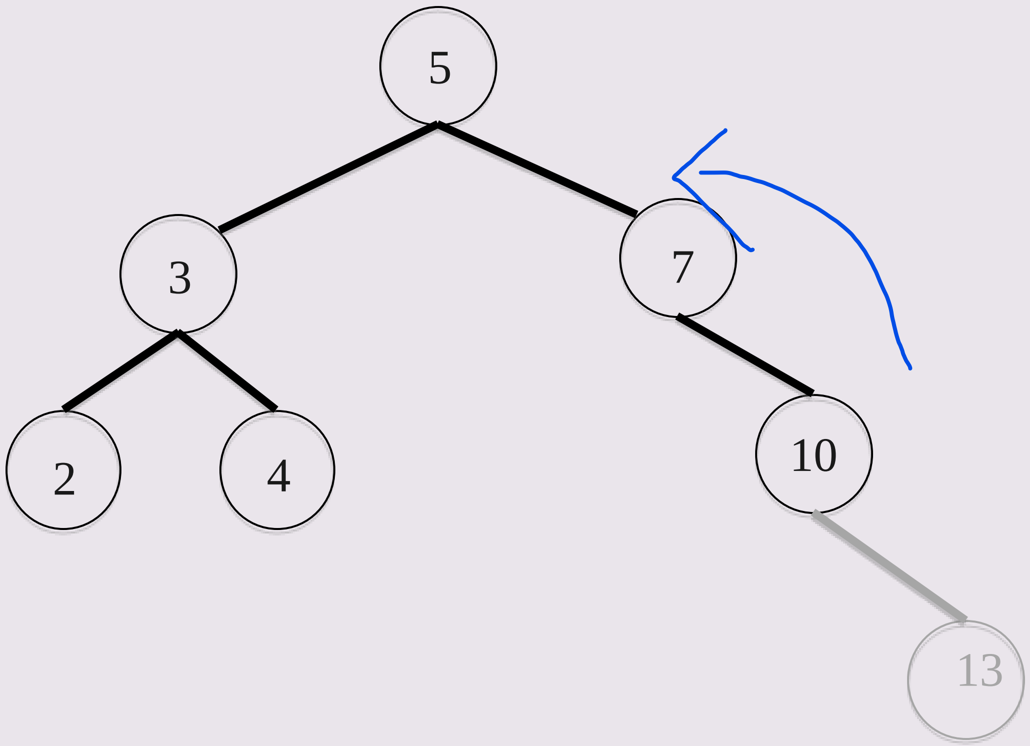
- How can we make it a AVL tree?



Restore AVL Property – example 1

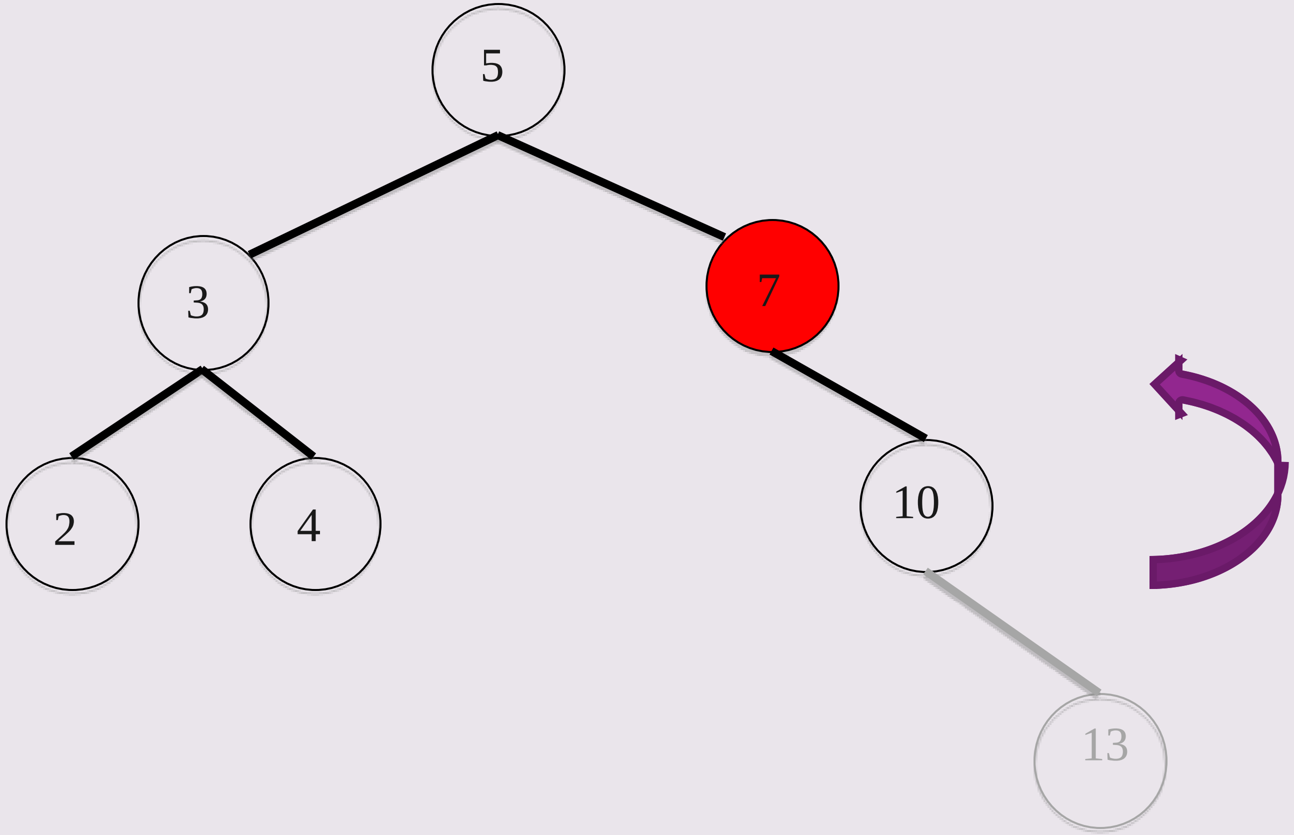
- What happens when we insert 13? Which node lose the AVL property?

7



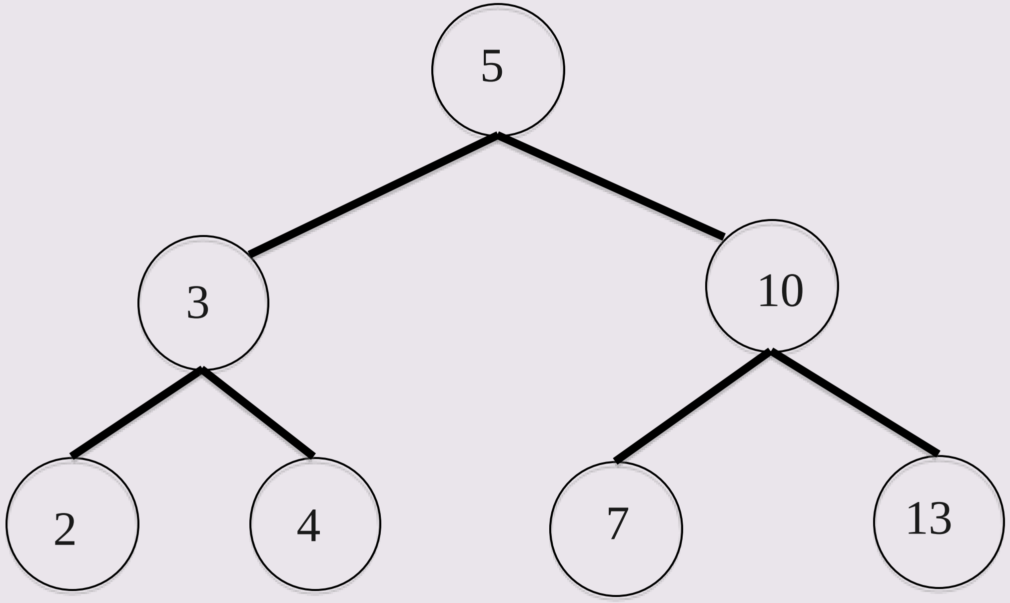
Restore AVL Property – example 1

- Rotation



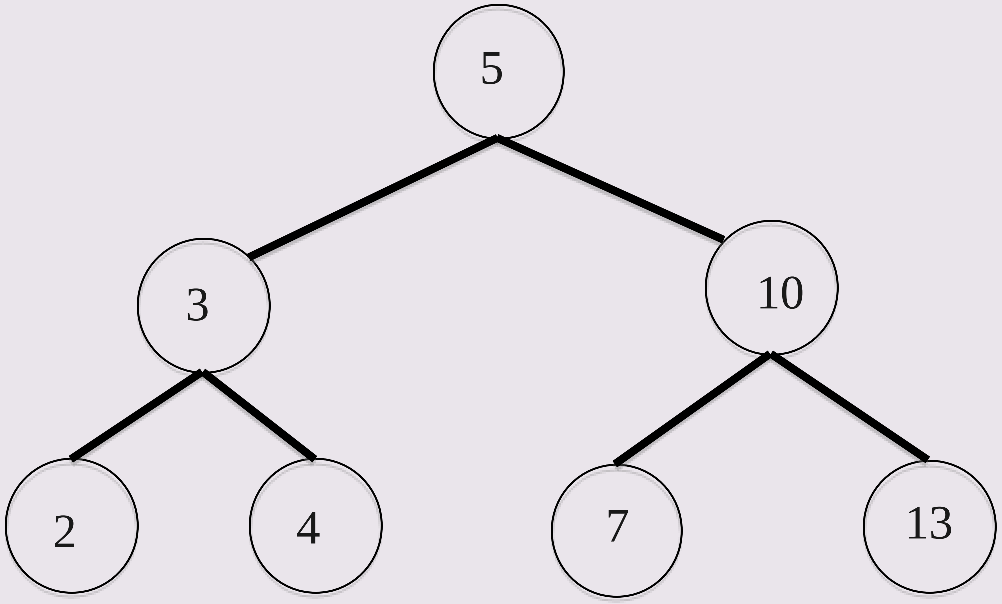
Restore AVL Property – example 1

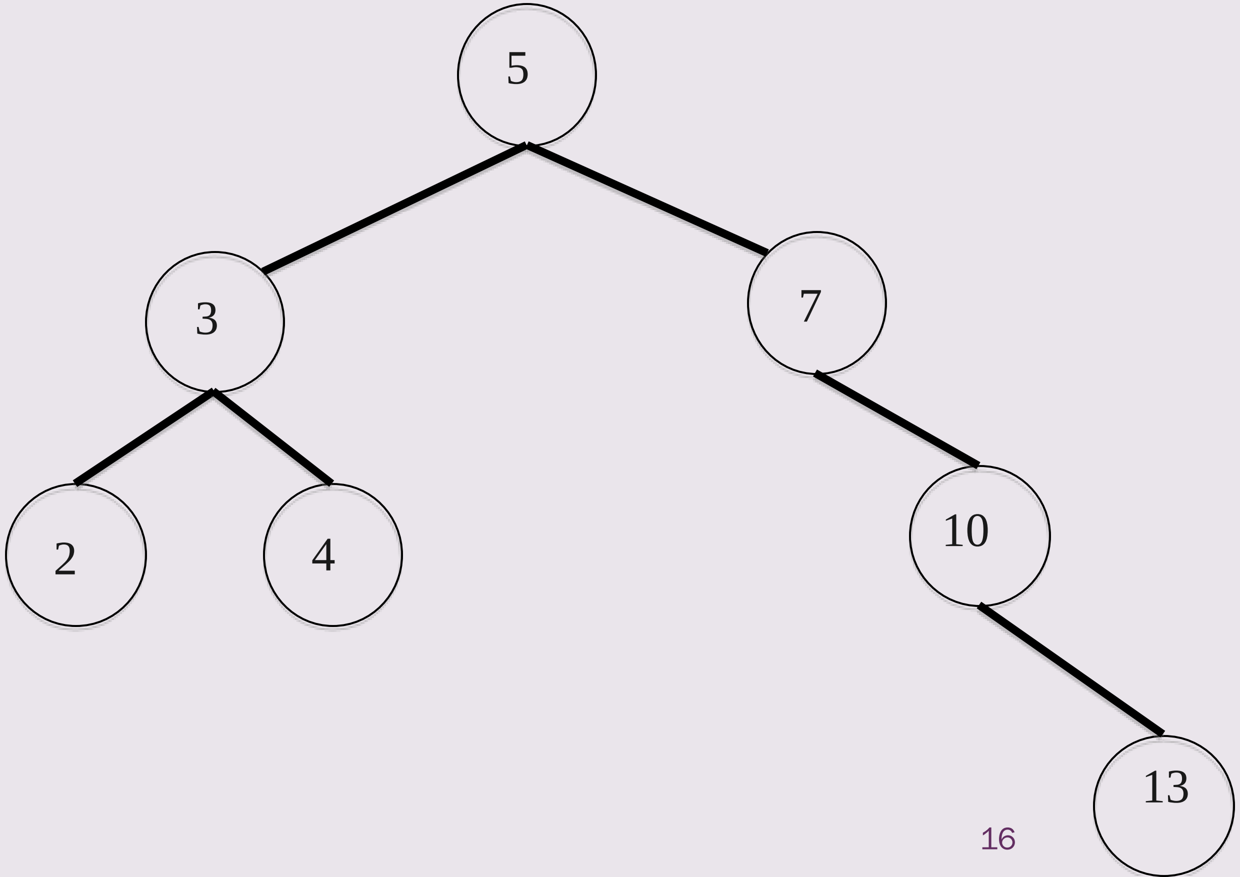
- Single Left Rotation



Restore AVL Property – example 1

- AVL tree V.S. Non-AVL tree

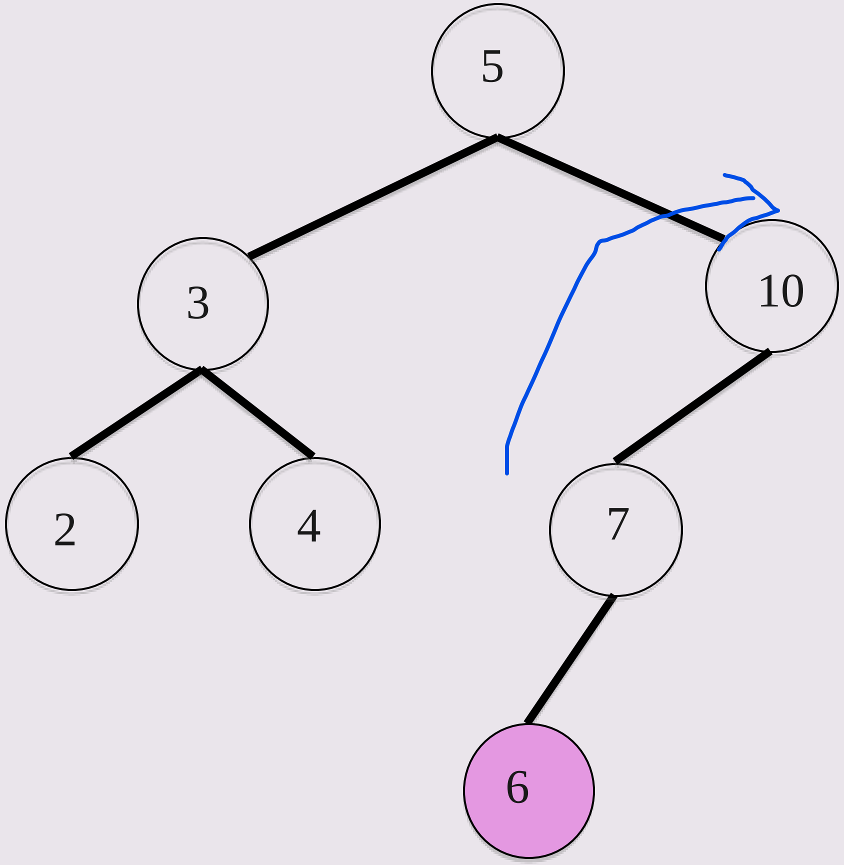




Restore AVL Property – example 2

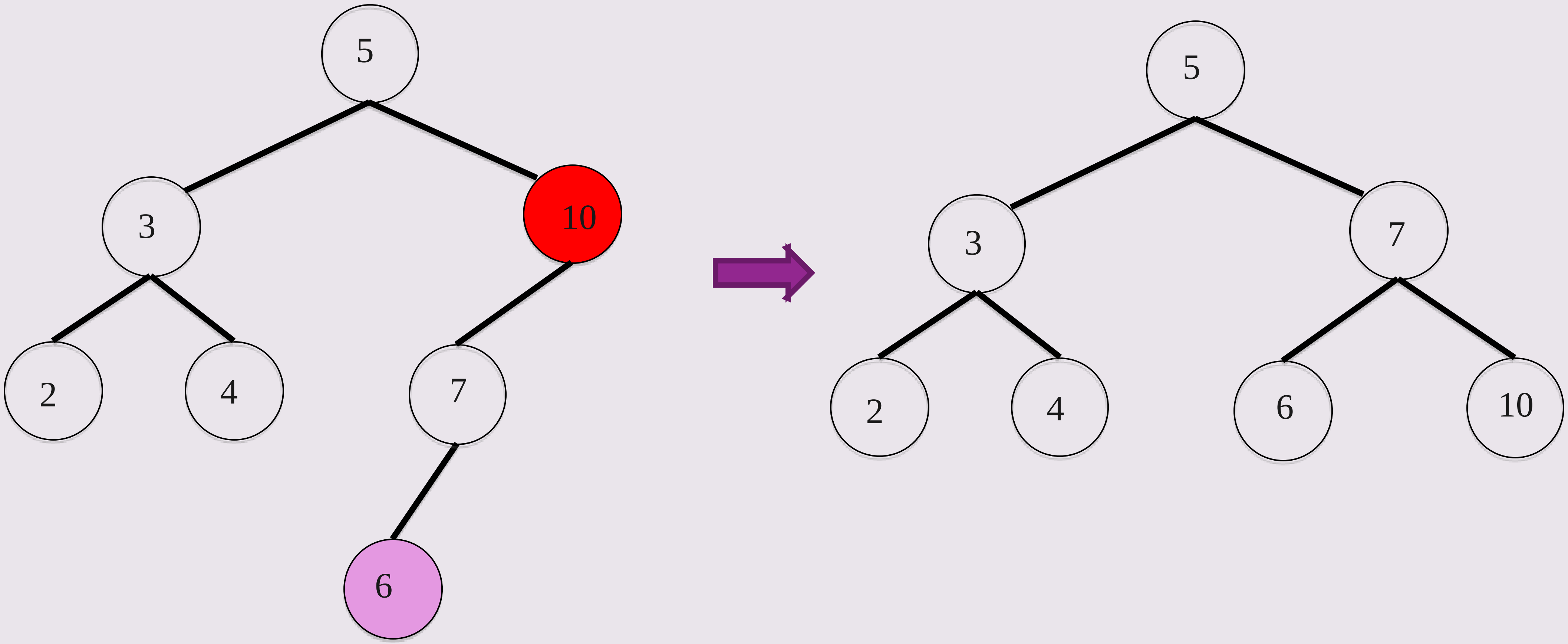
- Exercise Time!
- What about insert 6? Which node lose the AVL property?

10



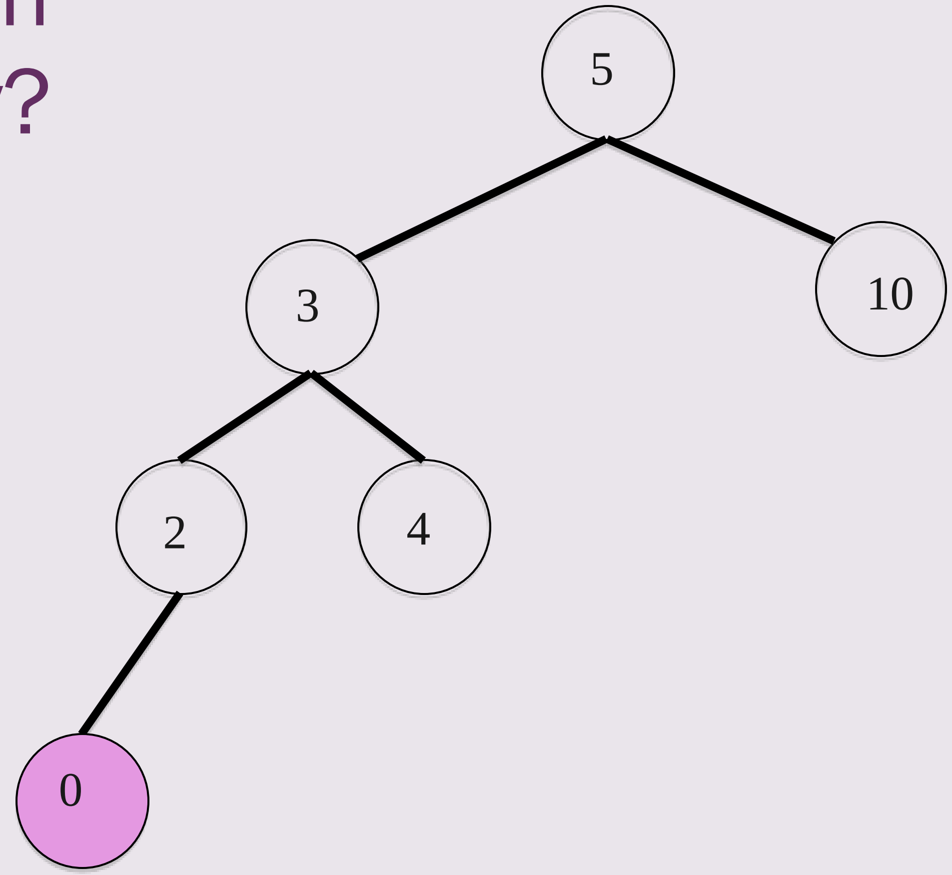
Restore AVL Property – example 2

- Single right rotation



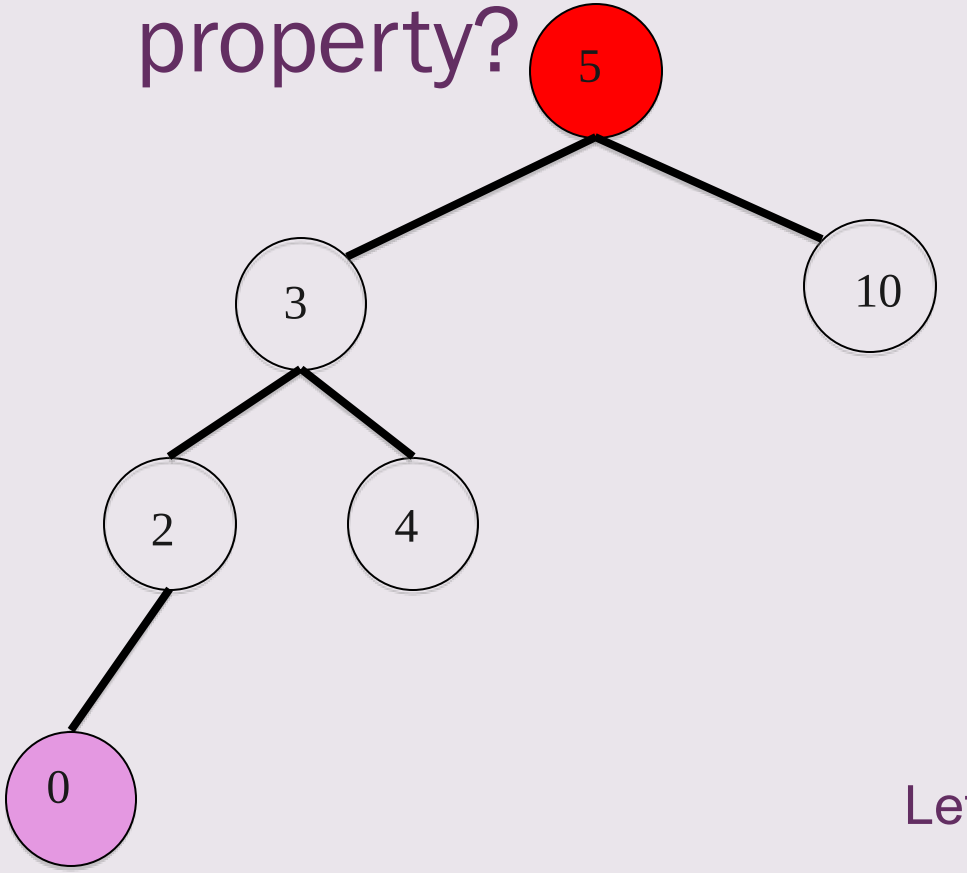
Restore AVL Property – example 3

- What about insert 0? Which node lose the AVL property?
- Discussion: Will previous rotation approach work?



Restore AVL Property – example 3

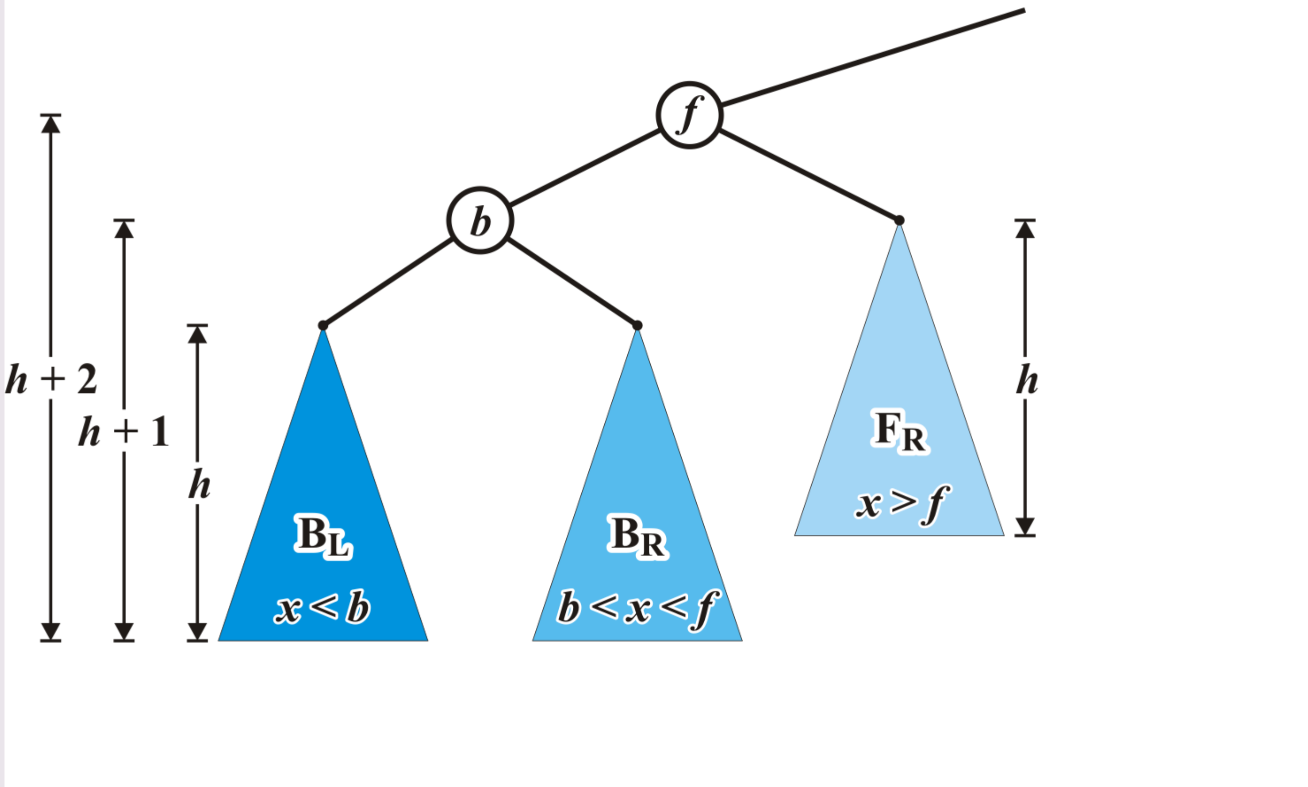
- What about insert 0? Which node lose the AVL property?



Let us look at things in the general case…

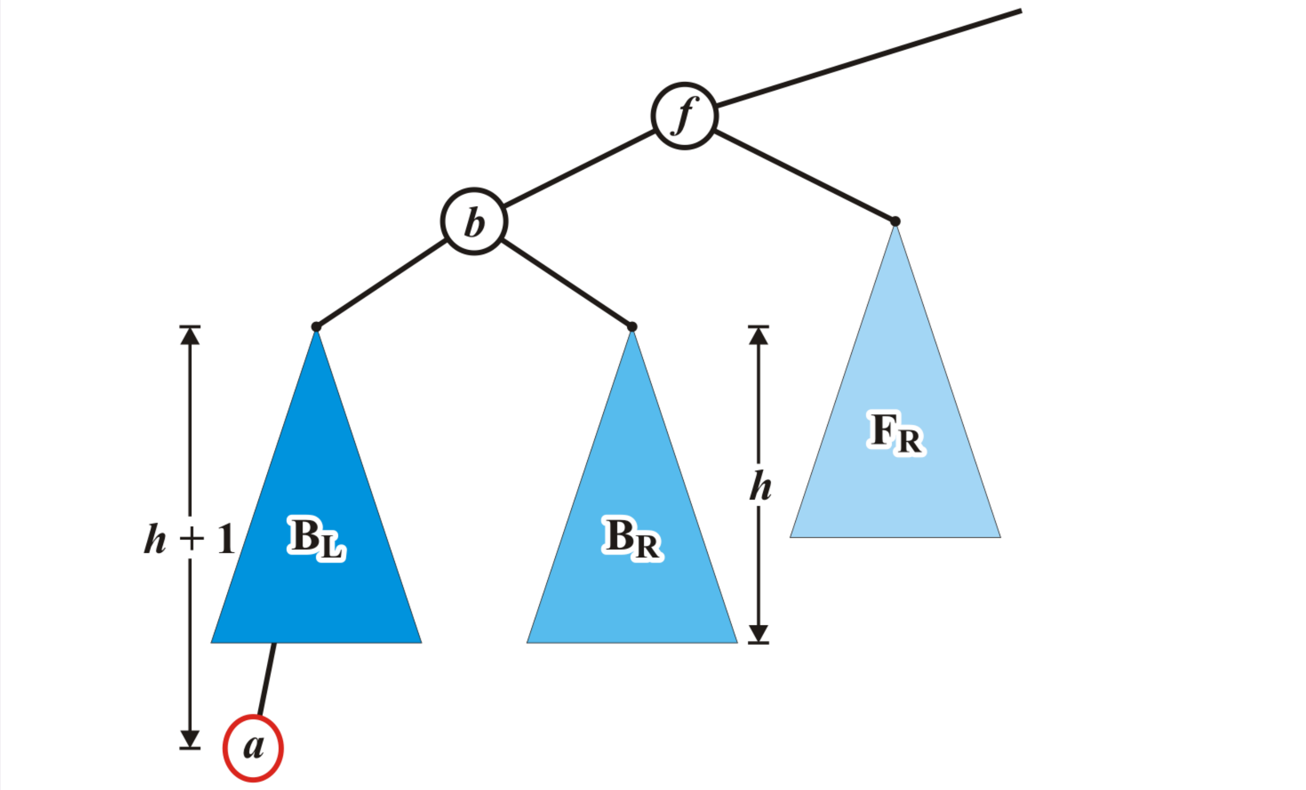
Restore AVL Property – General case 1

- Consider the following setup
  - *Each blue triangle represents a tree of height h*



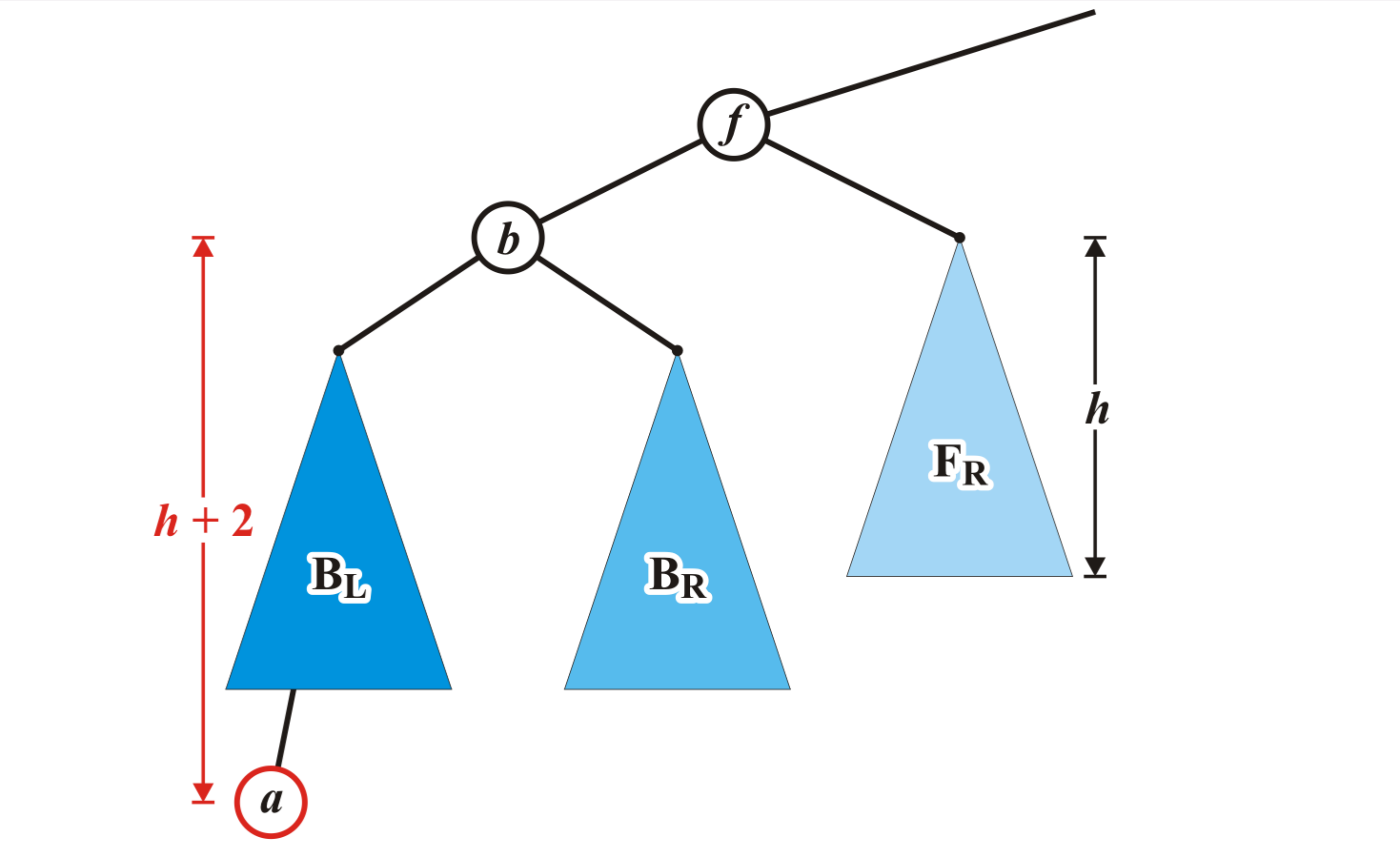
Restore AVL Property – General case 1

- Insert ***a*** into this tree: it falls into the left subtree $\mathbf{B_L}$ of ***b***



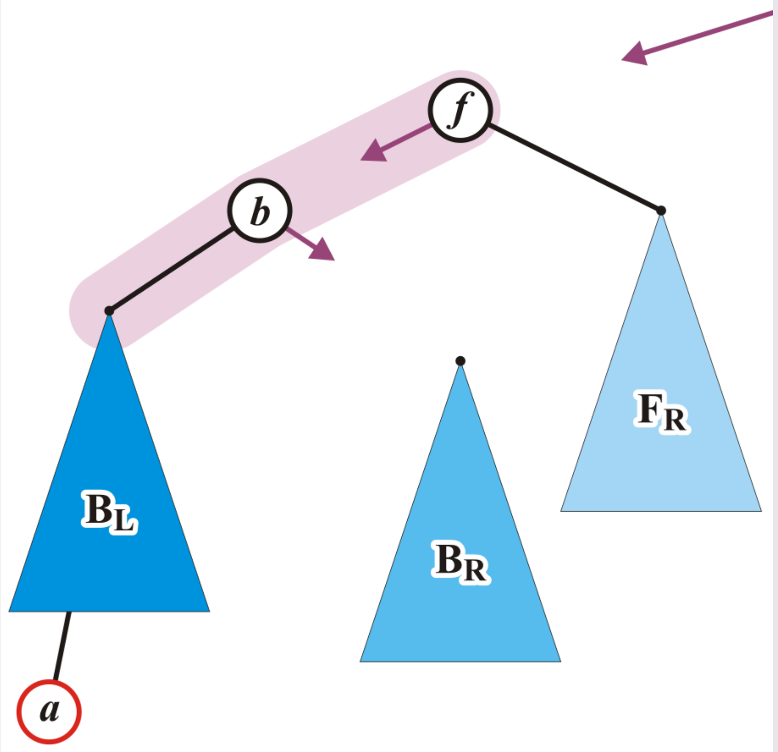
Restore AVL Property – General case 1

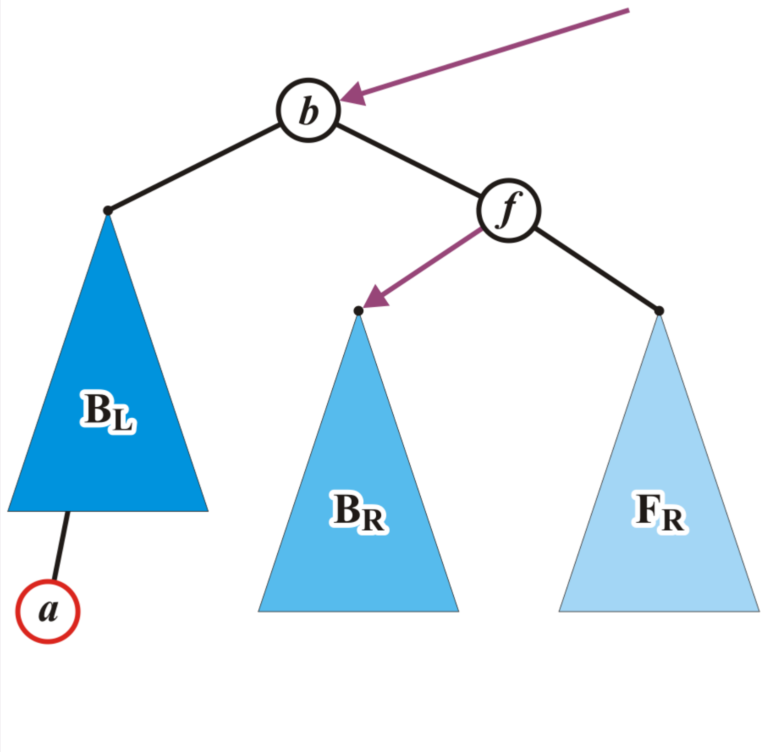
- Node *f* is now losing the AVL property



Restore AVL Property – General case 1

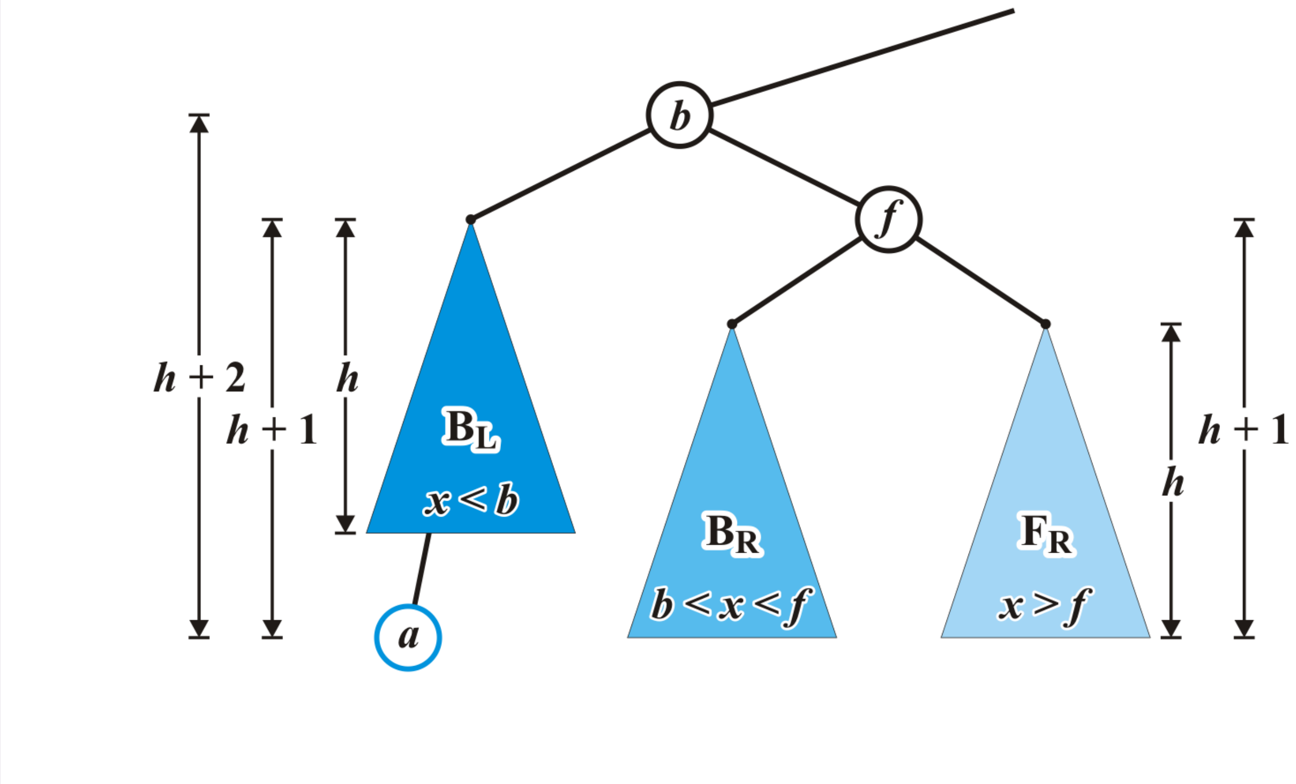
- Make node ***b*** to the root and denote node ***f*** to be the right child of ***b***
- Assign any former parent of node ***f*** to the address of node ***b***
- Assign the address of the tree $\mathbf{B_R}$ to be the left child of ***f***





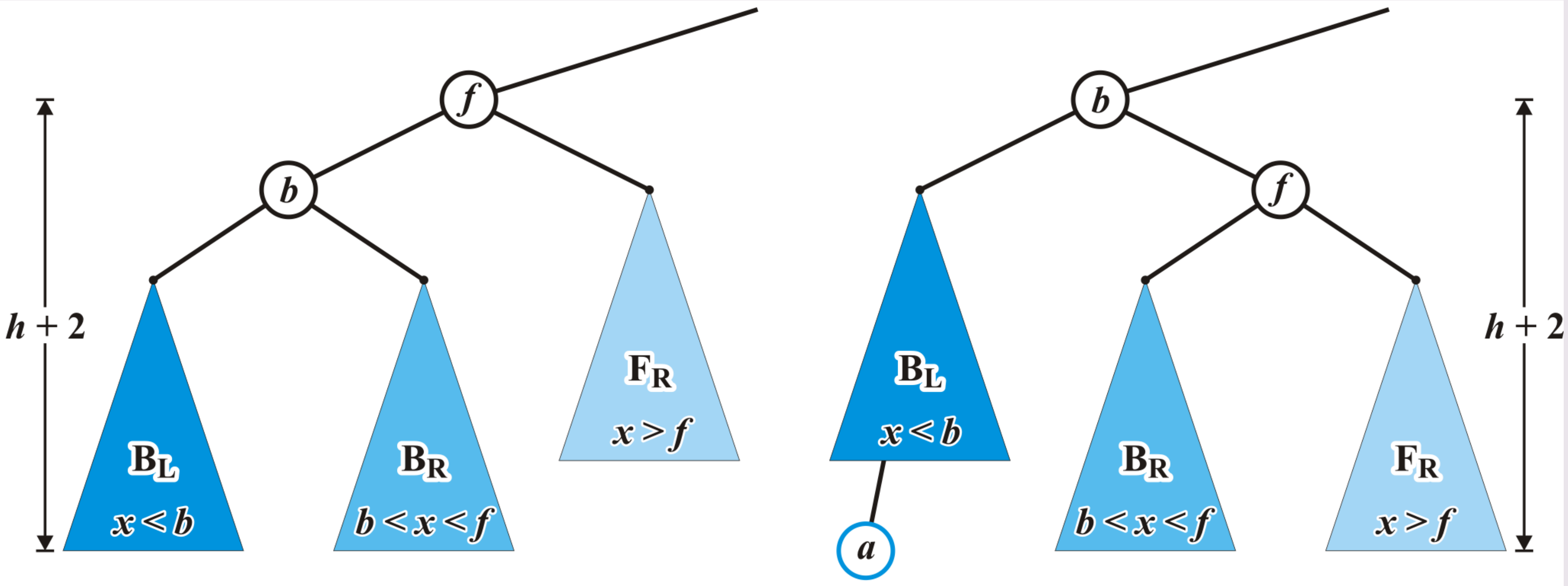
Restore AVL Property – General case 1

- The node b and f have AVL property
- Subtrees are in the correct position



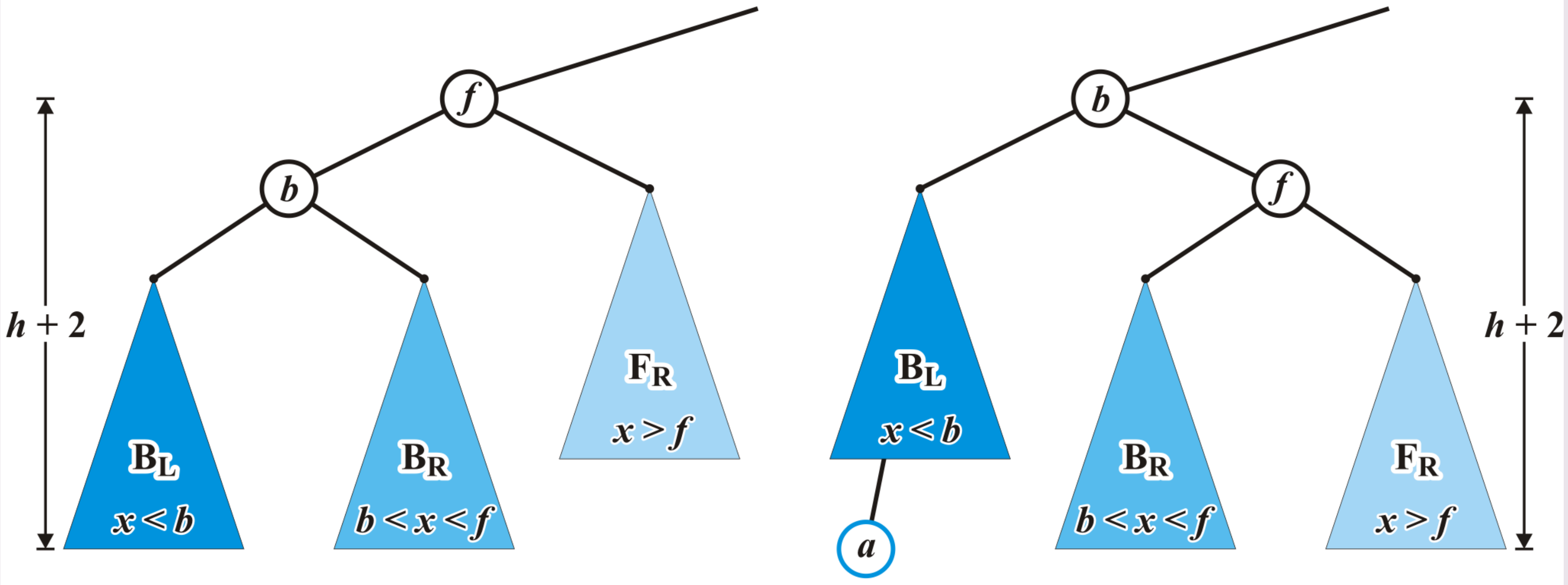
Restore AVL Property – General case 1

- Additionally, height of the corrected tree rooted at $\boldsymbol{b}$ equals the original height of the tree rooted at $\boldsymbol{f}$
  - *Thus, this insertion will no longer affect the balance of any ancestors all the way back to the root*



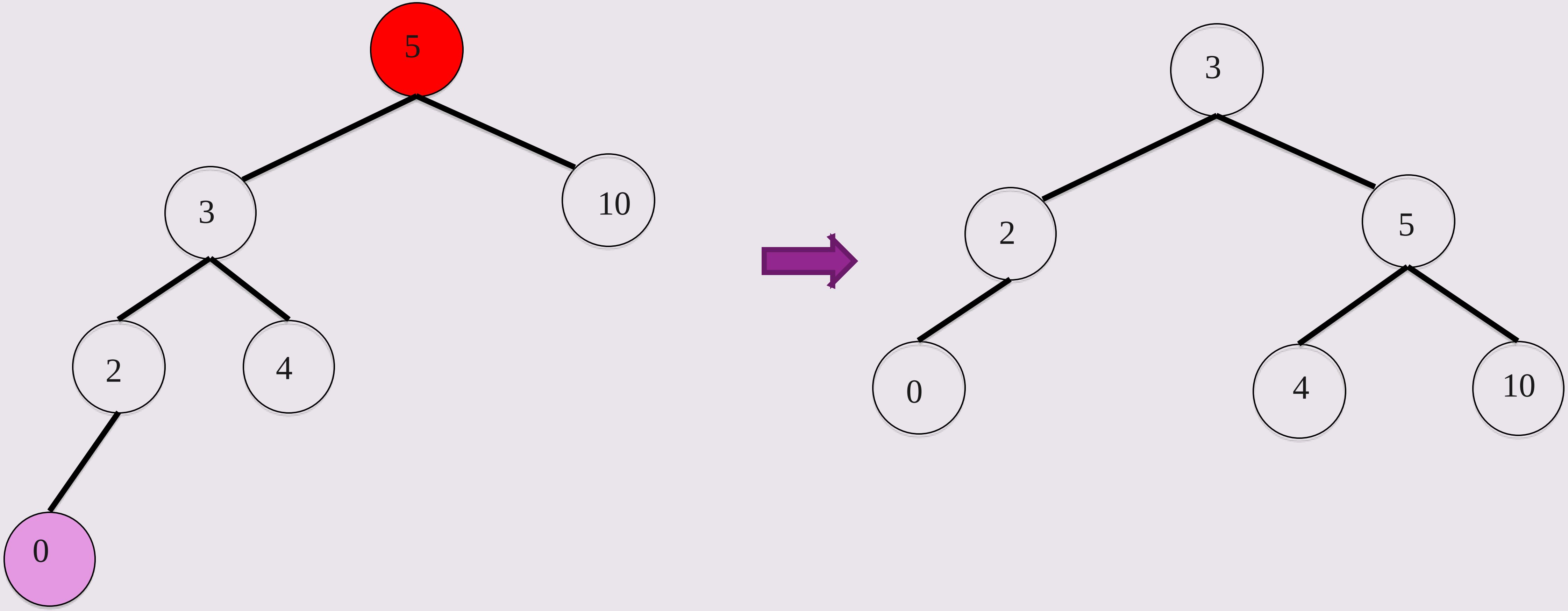
Restore AVL Property – General case 1

- This can be viewed as a general “right rotation”
- Excercise question: Can you describe the process of a general “left rotation”?



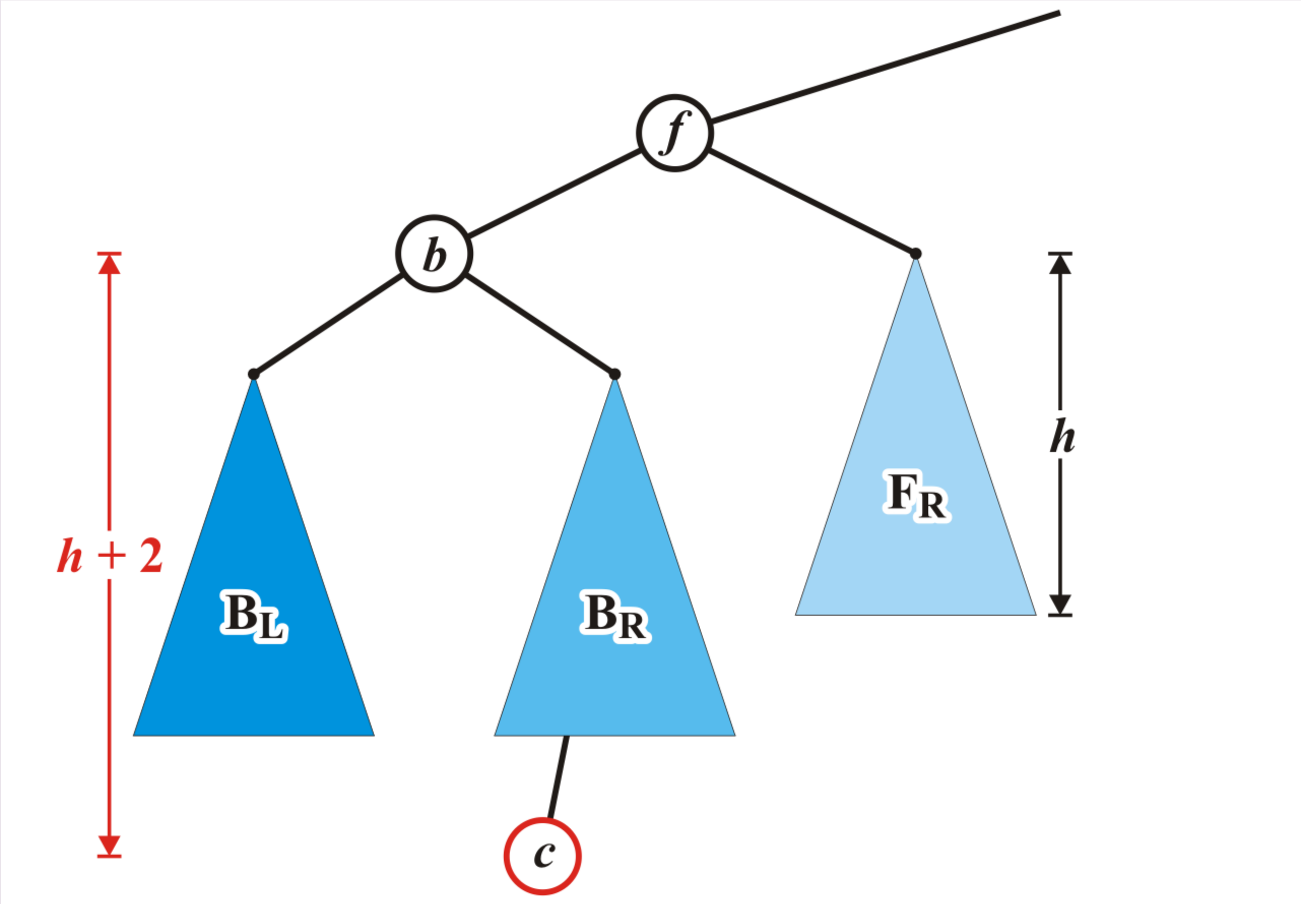
Restore AVL Property – General case 1

- Back to our example



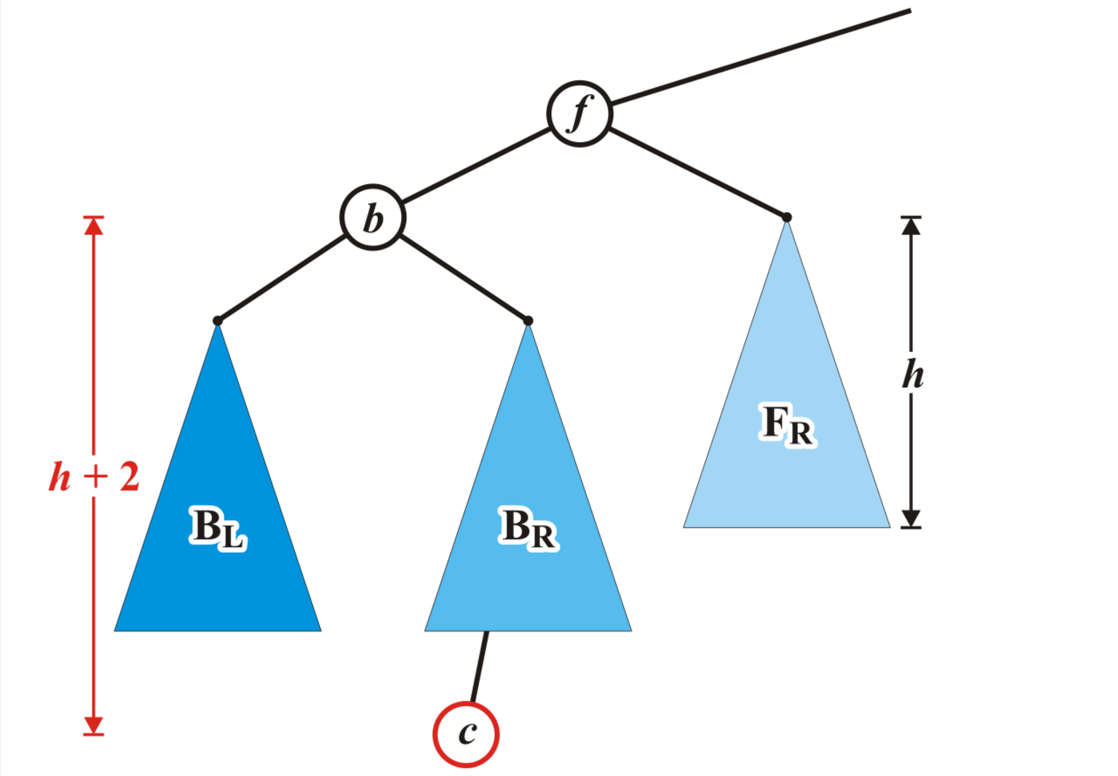
Restore AVL Property – General case 2

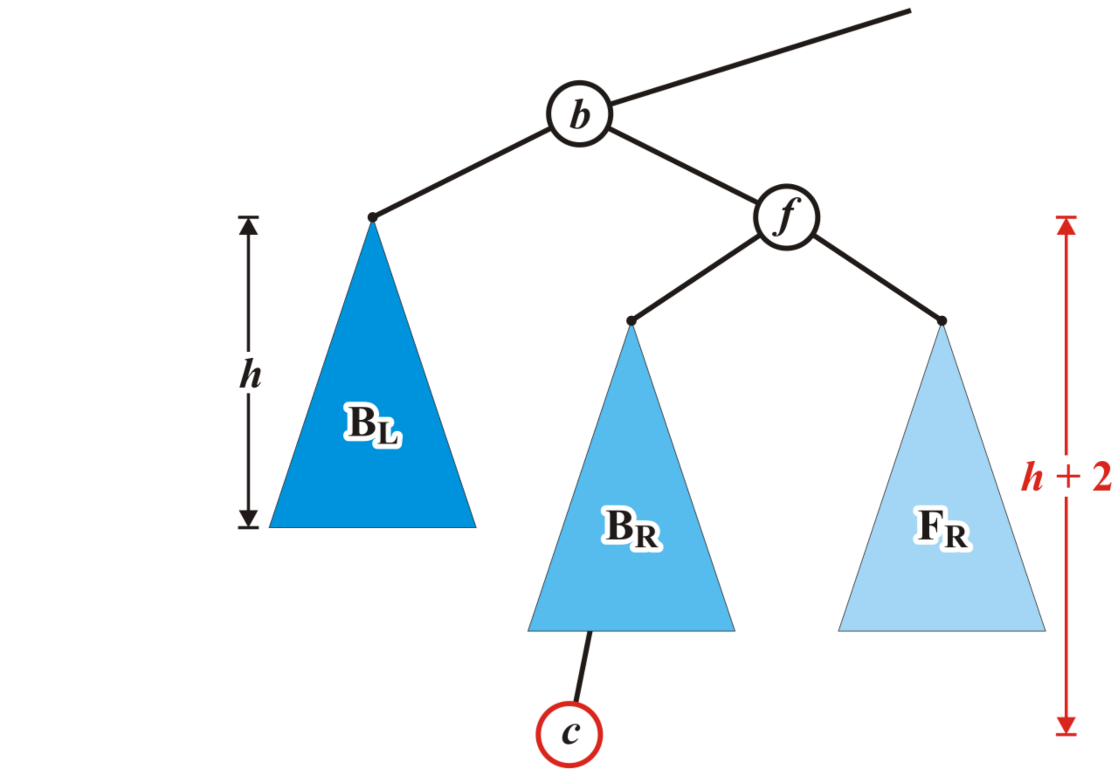
- Insertion of c where $b < c < f$ into our original tree



Restore AVL Property – General case 2

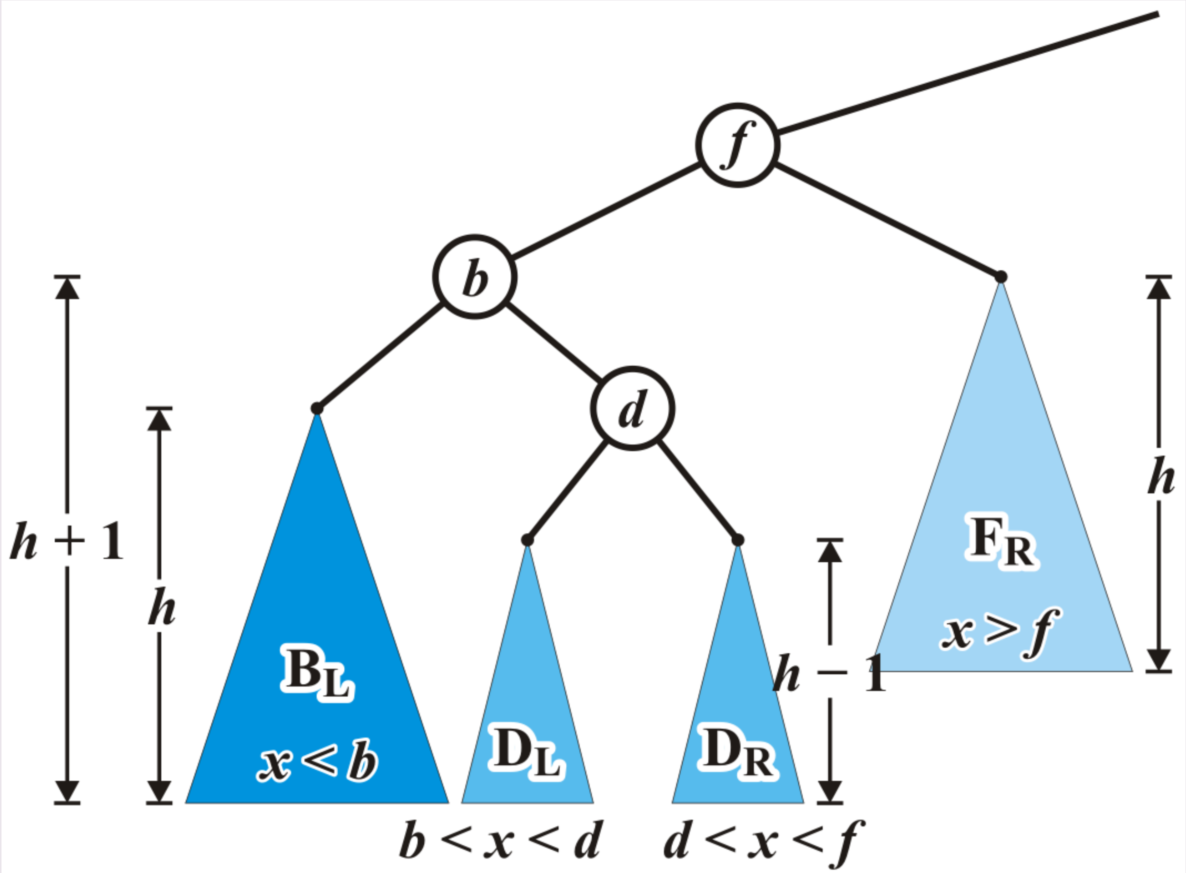
- The previous correction does not fix the imbalance at the root of this sub-tree: the new root, ***b***, remains unbalanced





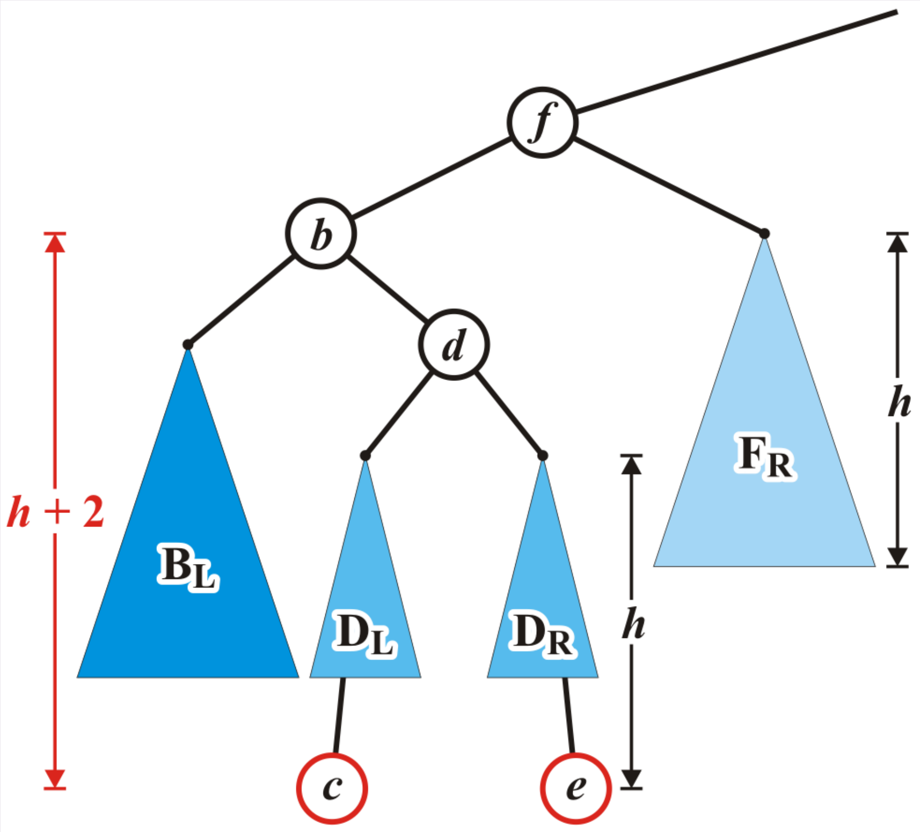
Restore AVL Property – General case 2

- Re-label the tree by dividing the left subtree of f into a tree rooted at d with two subtrees of height $h-1$



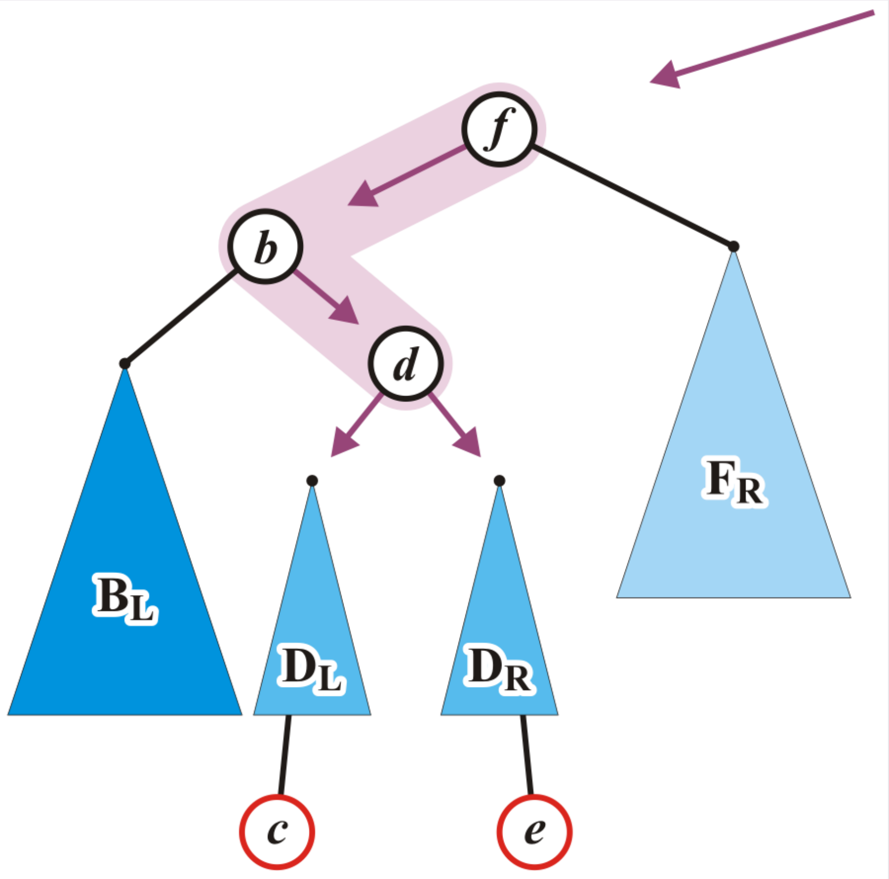
Restore AVL Property – General case 2

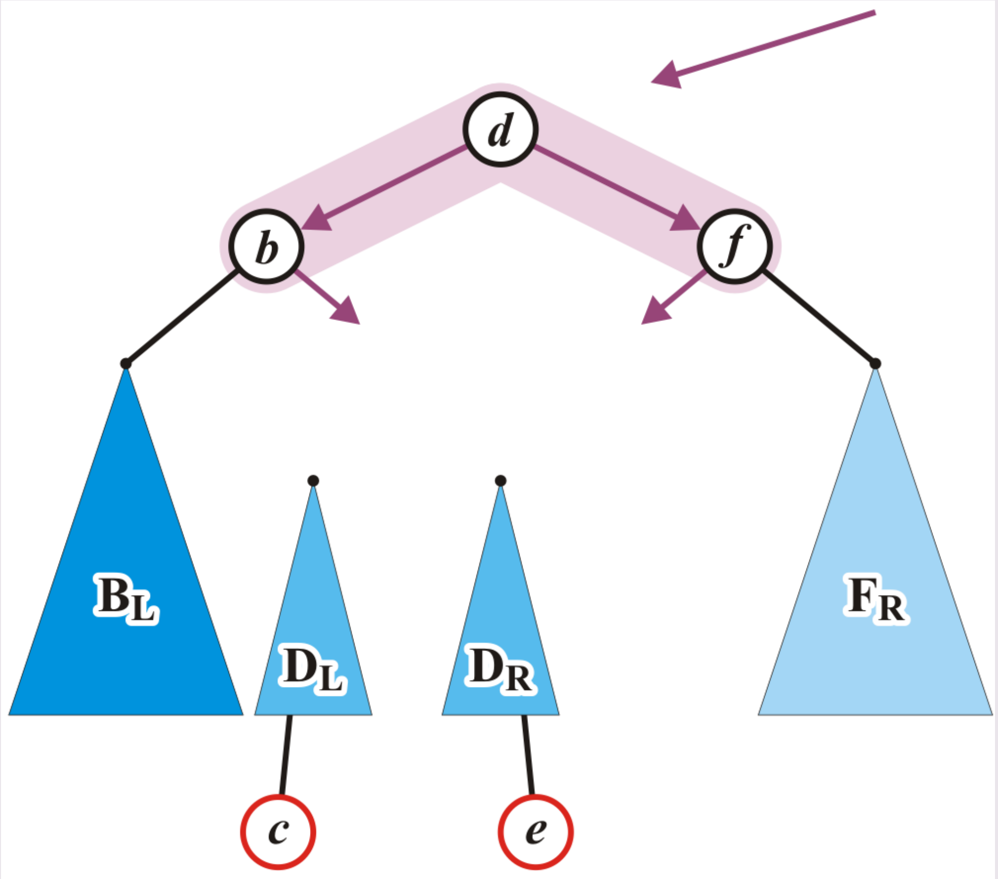
- Now an insertion causes an imbalance at *f*
  - *The addition of either* **c** *or* **e** *will cause this*



Restore AVL Property – General case 2

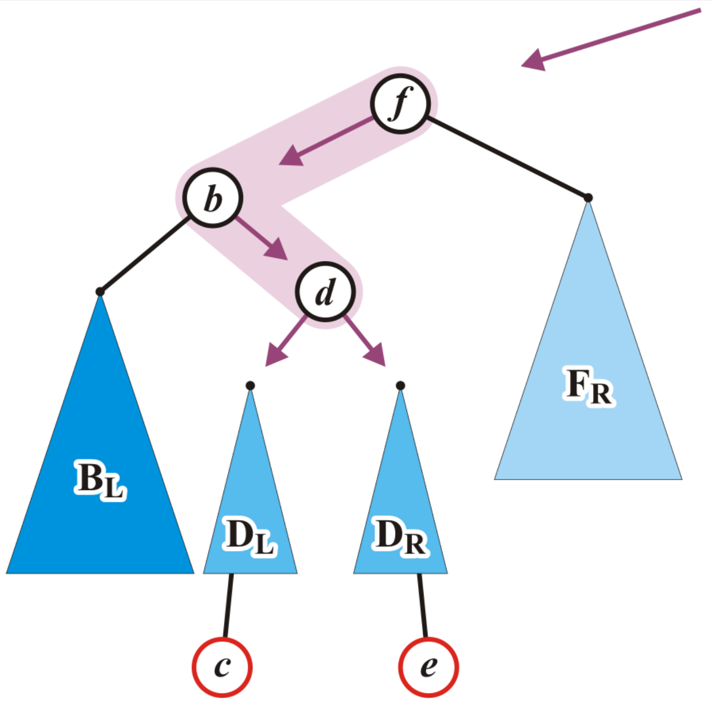
- ***b*** and ***f*** will be assigned as children of the new root ***d***

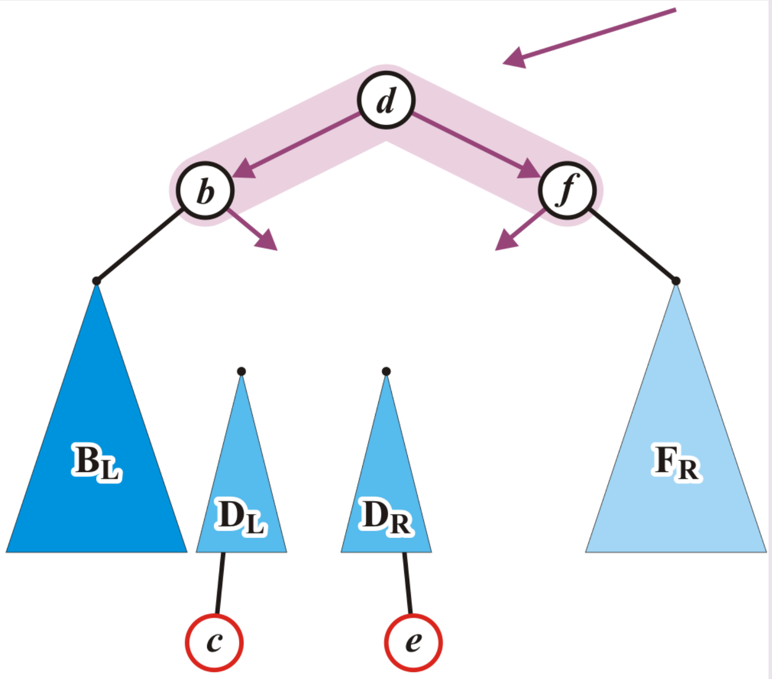




Restore AVL Property – General case 2

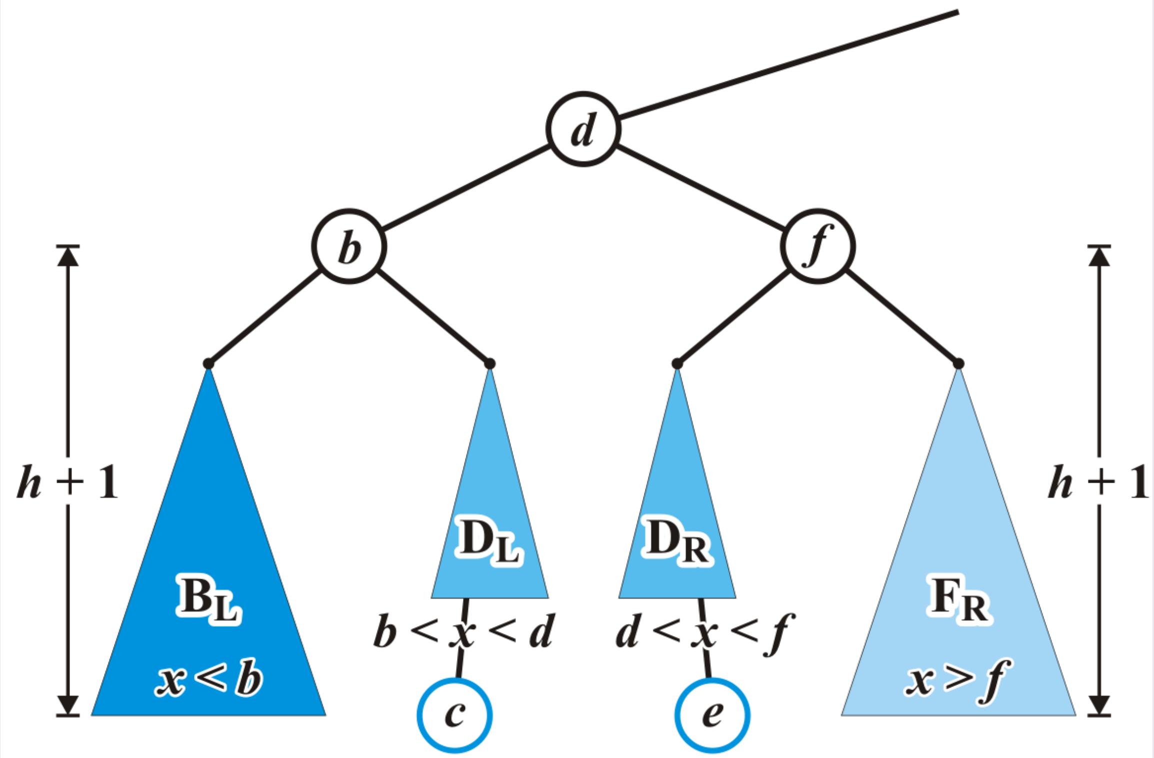
- Some view it as a Left-Right rotation
  - ***b* and *d* – *left rotation***
  - ***d* and *f* – *right rotation***





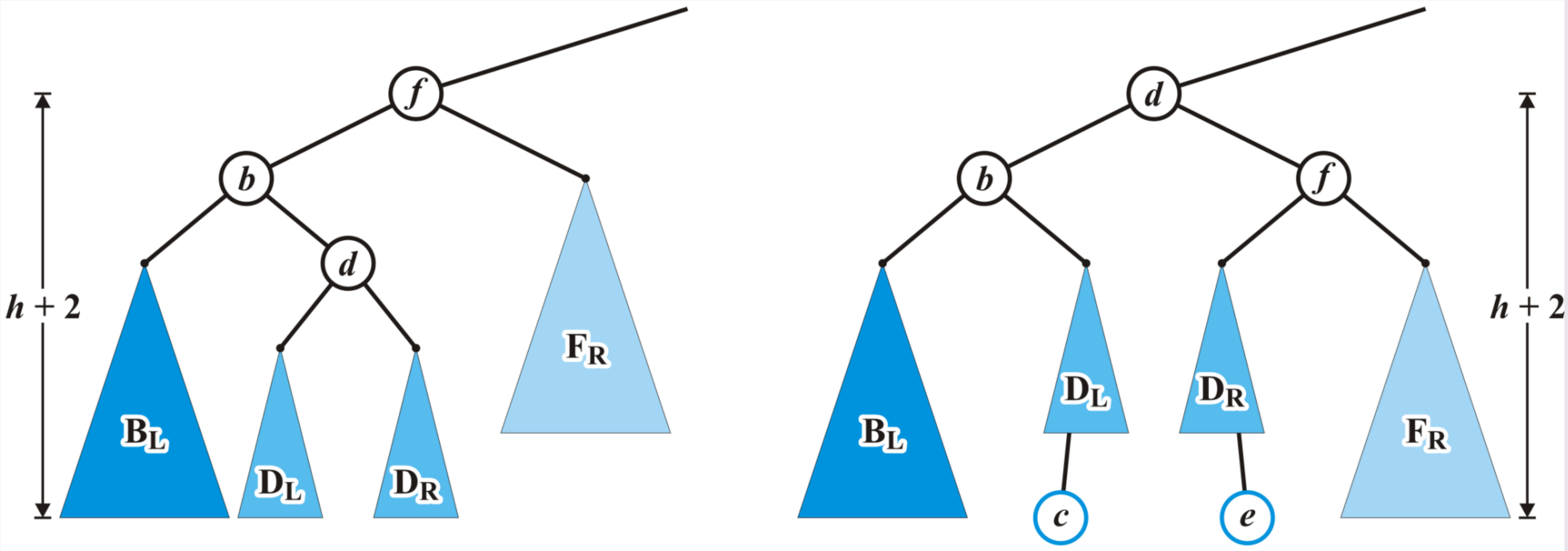
Restore AVL Property – General case 2

- Now the tree rooted at ***d*** is balanced
  - *After the correction, height of **b** and **f** become h + 1 and **d** is h + 2*



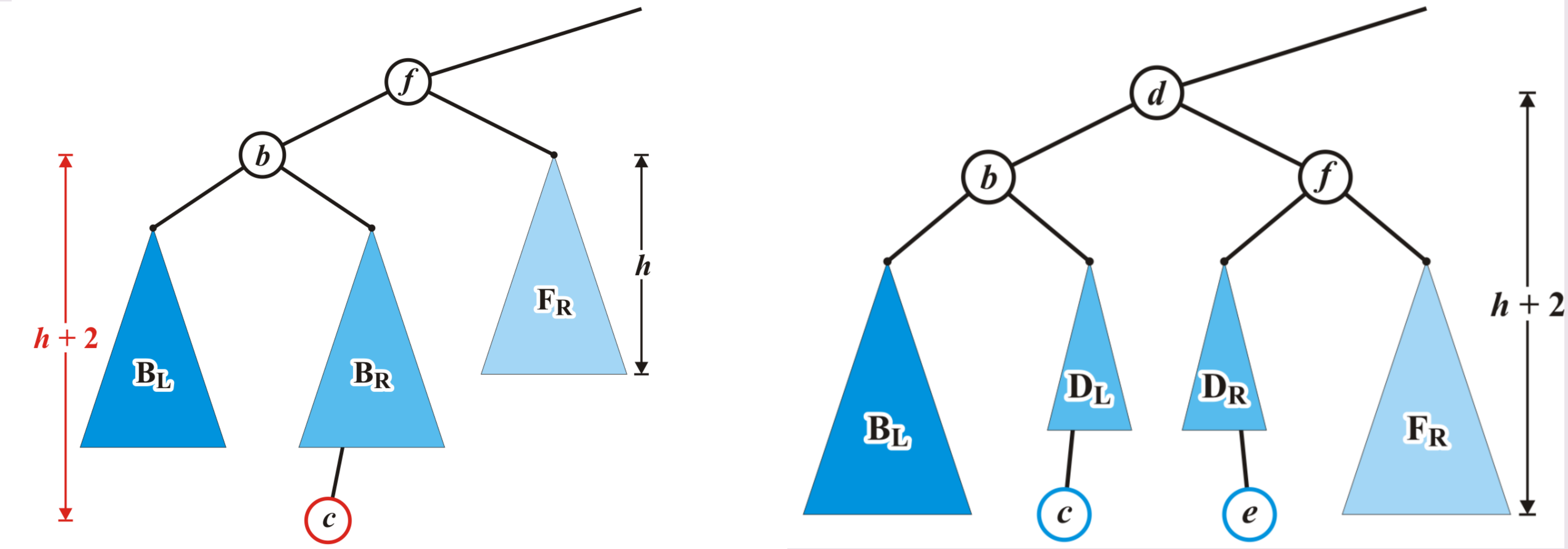
Restore AVL Property – General case 2

- The height of the root did not change



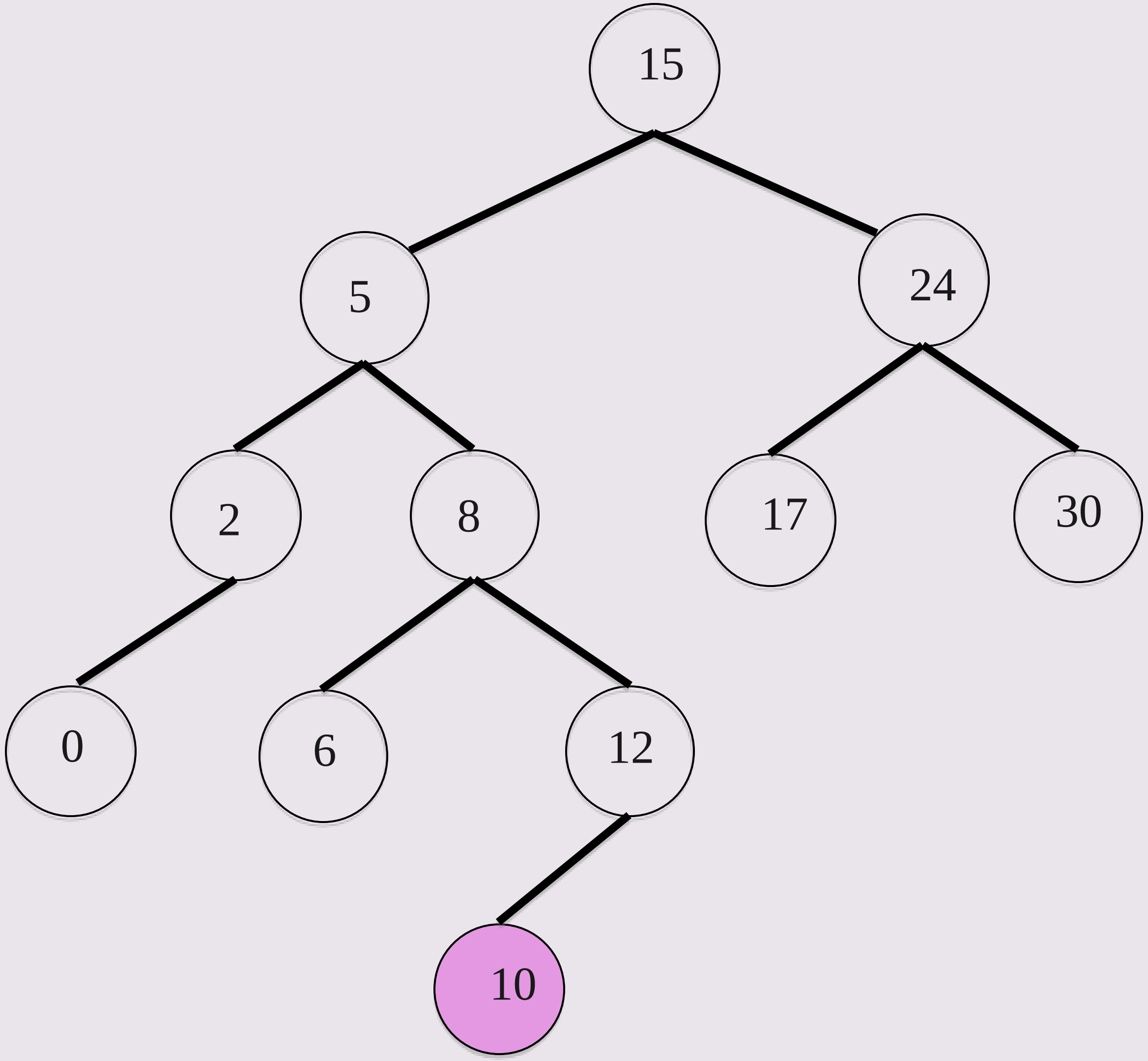
Restore AVL Property – General case 2

- This can be viewed as a “Left-Right rotation”
- Excercise question: Can you describe the process of a “Right-Left rotation”?



Exercise

- Can you make the following to be an AVL tree?
  - *Hint: general case 2*



Summary

- AVL tree definition and examples
- The reasons of using AVL trees
- Insert nodes into the tree and restore AVL property
  - *Simple right/left rotation*
  - *Two general cases*

References

- Book “Thomas A. Standish: Data Structures, Algorithms & Software Principles in C, Addison Wesley”
- Teaching notes by Douglas Wilhelm Harder, University of Waterloo
- Teaching notes by Lingling Jin, Thompson Rivers University

ANY QUESTIONS?