CSE 332 Summer 2024
Lecture 8: AVL Trees

Nathan Brunelle

http://www.cs.uw.edu/332

Dictionary (Map) ADT

- Contents:
  - Sets of key+value pairs
  - Keys must be comparable
- Operations:
  - insert(key, value)
    - Adds the (key,value) pair into the dictionary
    - If the key already has a value, overwrite the old value
      - Consequence: Keys cannot be repeated
  - find(key)
    - Returns the value associated with the given key
  - delete(key)
    - Remove the key (and its associated value)

Naïve attempts

Data Structure	Time to insert	Time to find	Time to delete
Unsorted Array	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$
Unsorted Linked List	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$
Sorted Array	$\Theta(n)$	$\Theta(\log n)$	$\Theta(n)$
Sorted Linked List	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
Heap	$\Theta(\log n)$	$\Theta(n)$	$\Theta(n)$
Binary Search Tree	$\Theta(\text{height})$	$\Theta(\text{height})$	$\Theta(\text{height})$
AVL Tree	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$

Improving the worst case

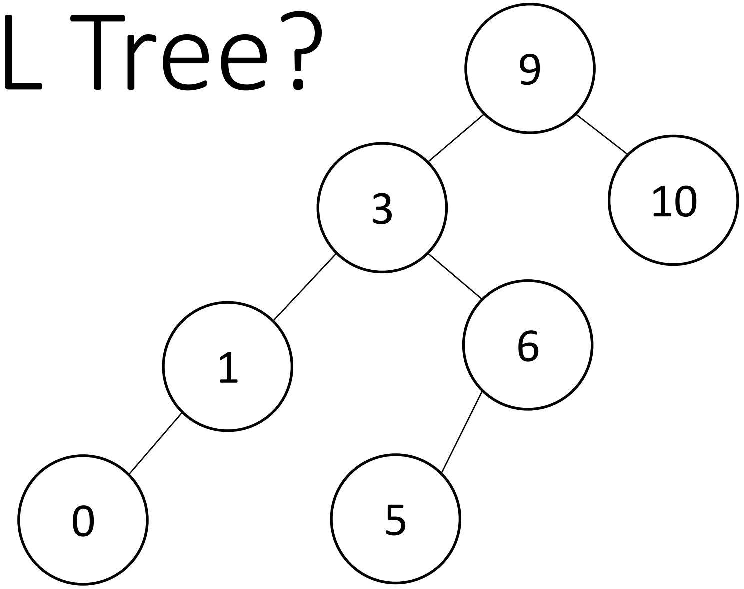
- How can we get a better worst case running time?
    - Add rules about the shape of our BST
- AVL Tree
    - A BST with some shape rules
        - Algorithms need to change to accommodate those

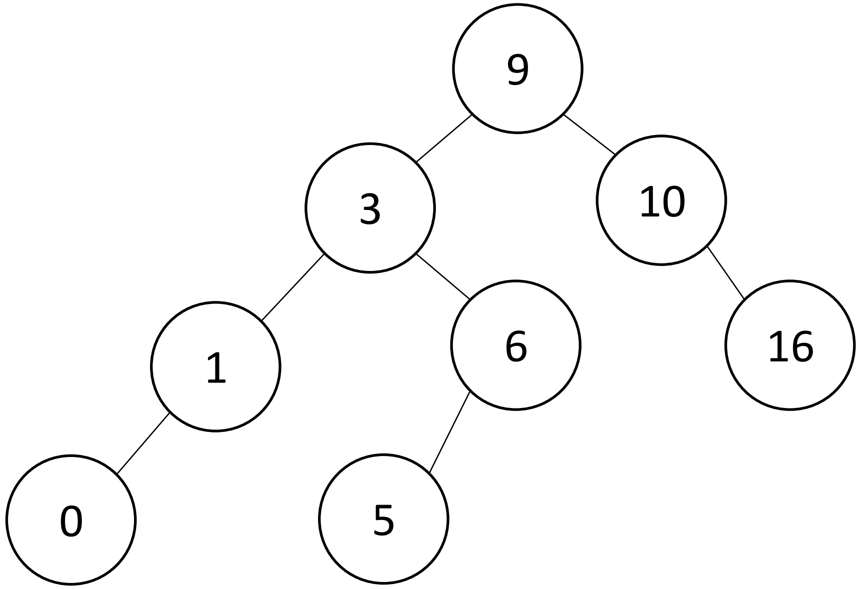
AVL Tree

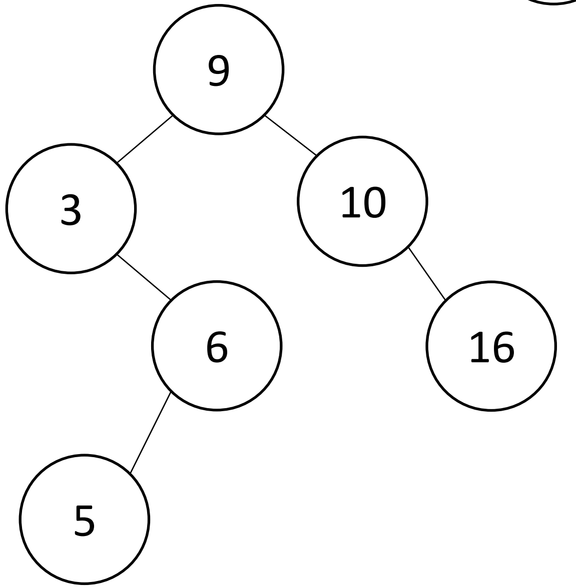
- A Binary Search tree that maintains that the left and right subtrees of every node have heights that differ by at most one.
  - height of left subtree and height of right subtree off by at most 1
  - Not too weak (ensures trees are short)
  - Not too strong (works for any number of nodes)

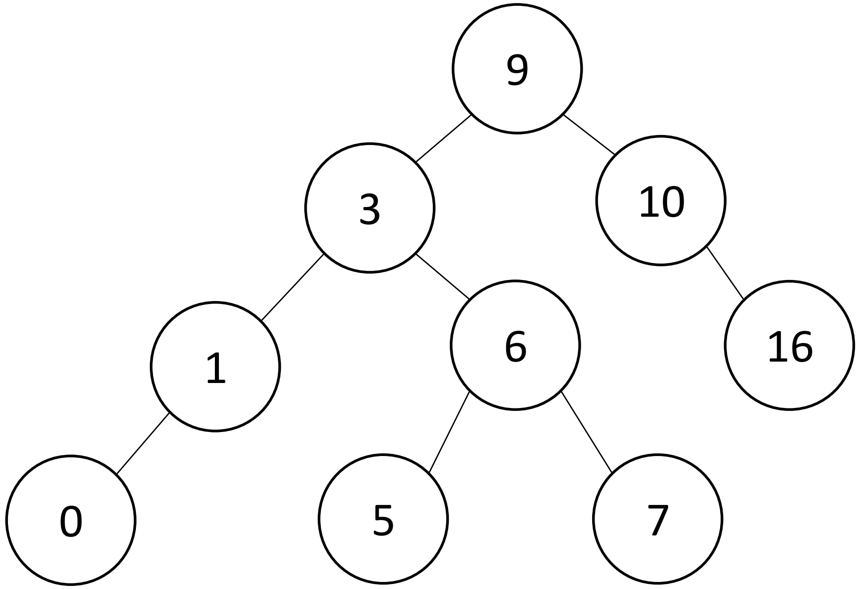
- Idea of AVL Tree:
  - When you insert/delete nodes, if tree is “out of balance” then modify the tree
  - Modification = “rotation”

Is it an AVL Tree?



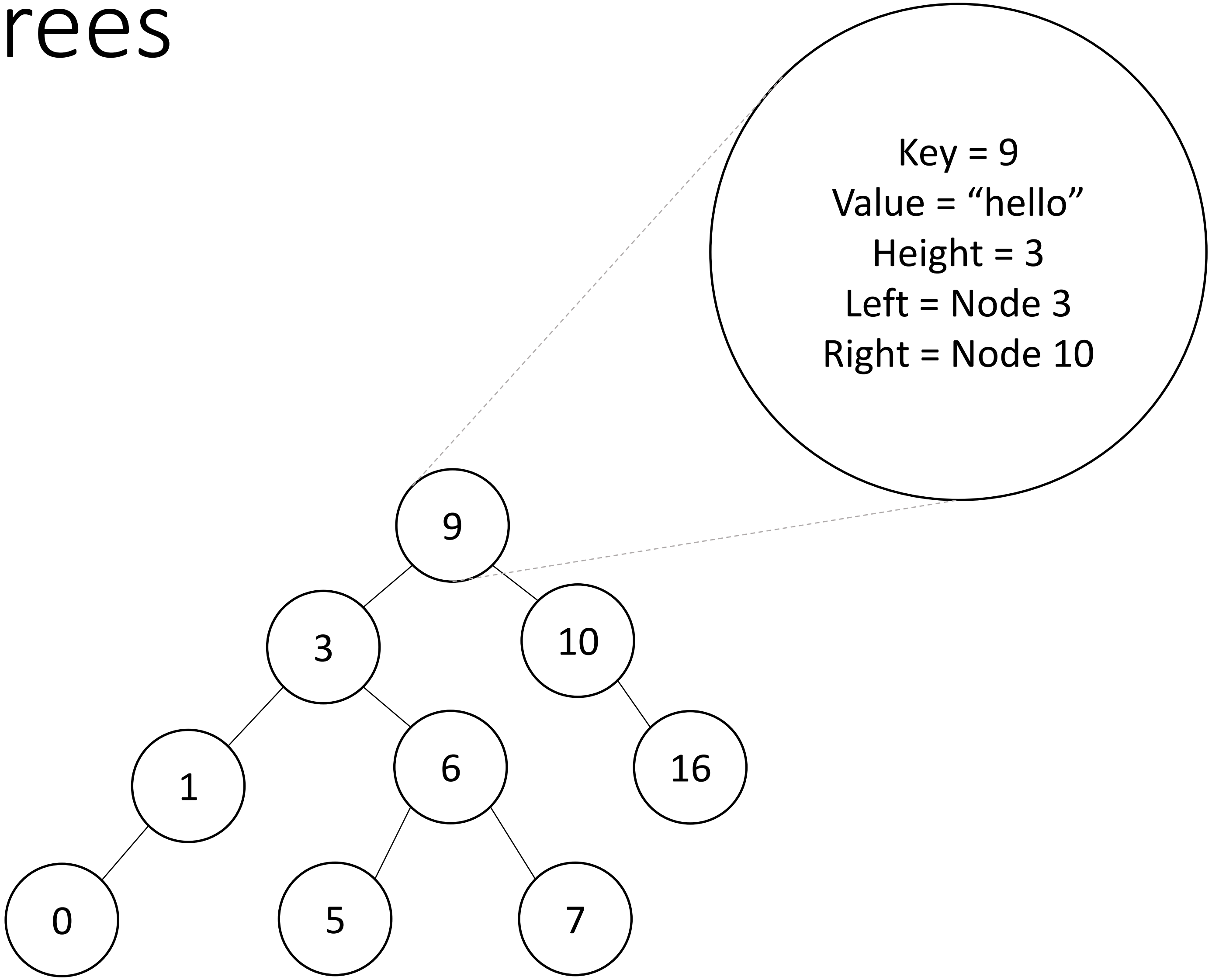






Using AVL Trees

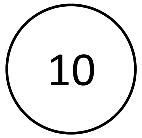
- Each node has:
  - Key
  - Value
  - Height
  - Left child
  - Right child

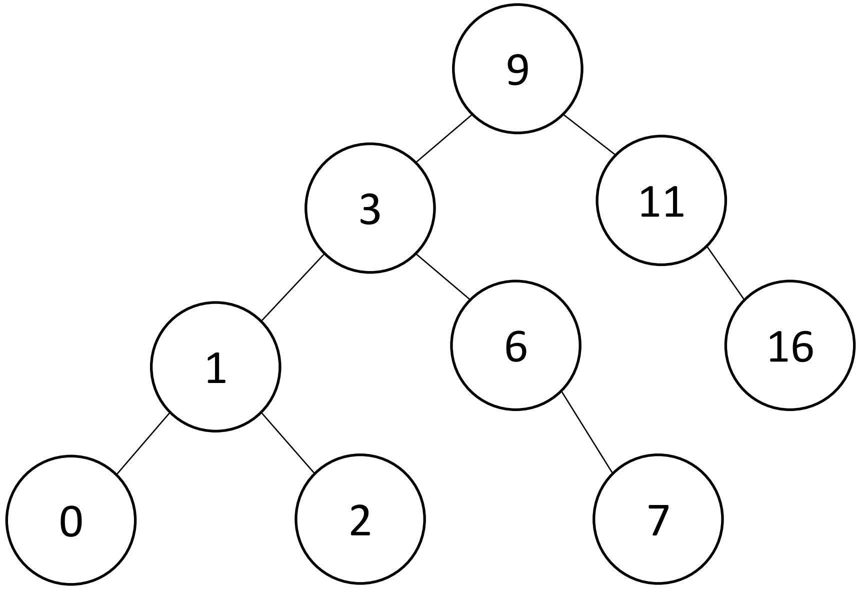


Inserting into an AVL Tree

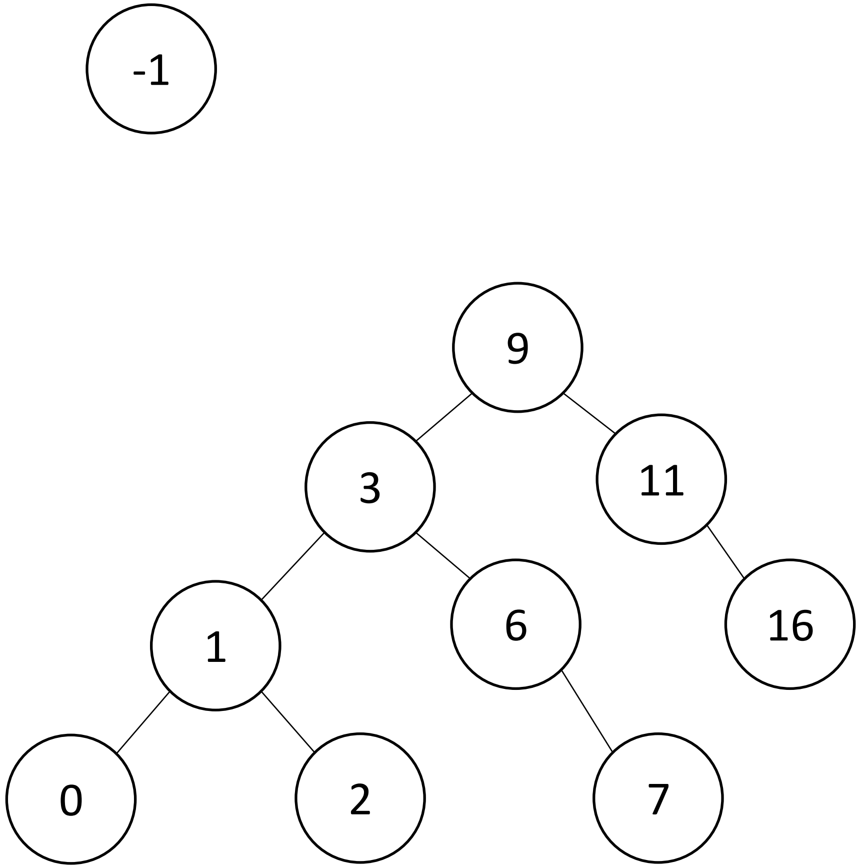
- Starts out the same way as BST:
  - “Find” where the new node should go
  - Put it in the right place (it will be a leaf)
- Next check the balance
  - If the tree is still balanced, you’re done!
  - Otherwise we need to do rotations

Insert Example

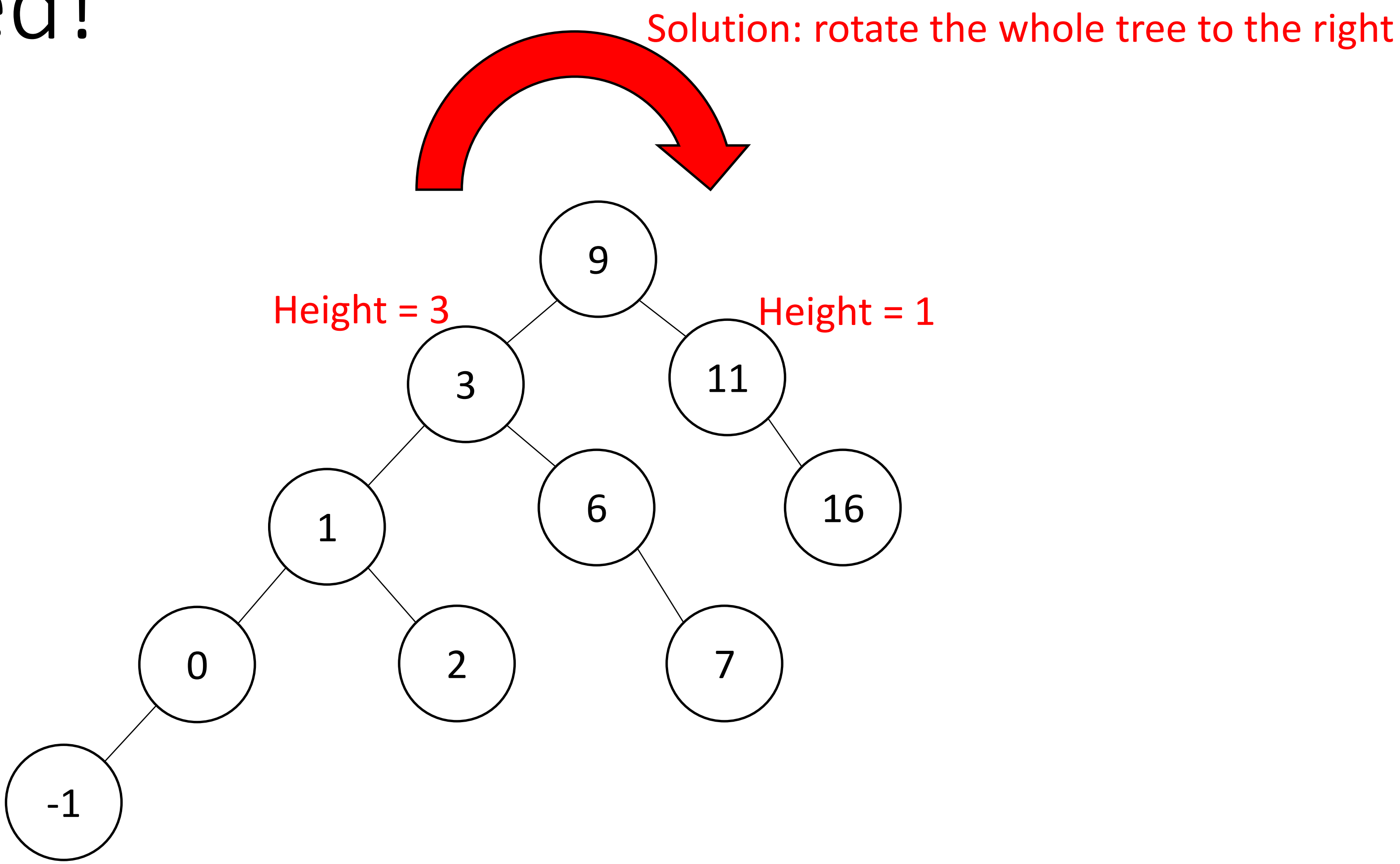


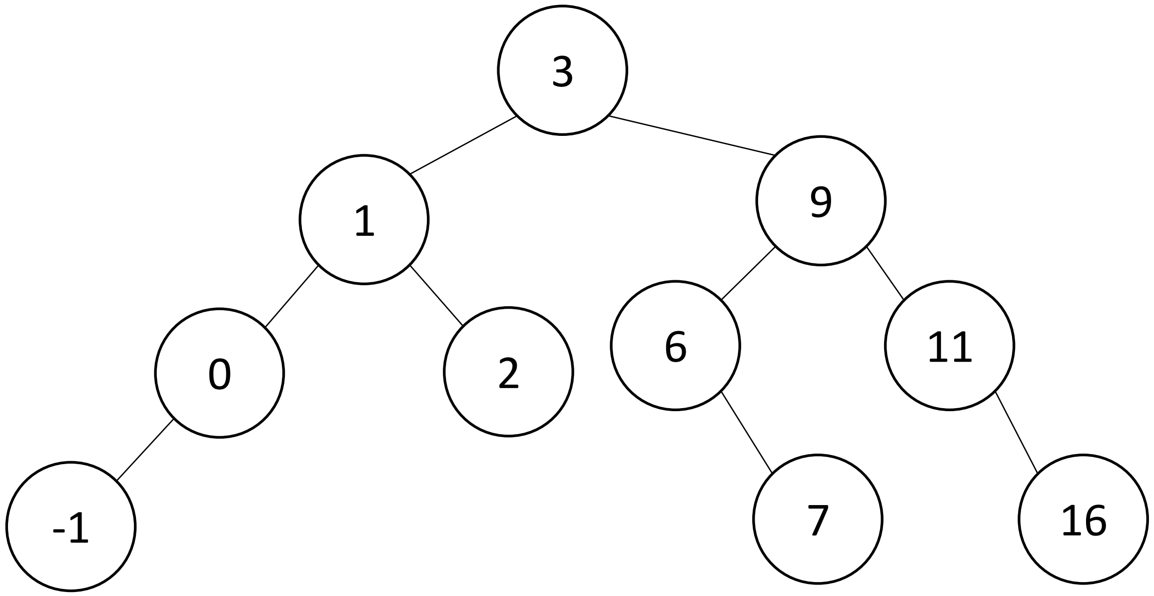


Insert Example

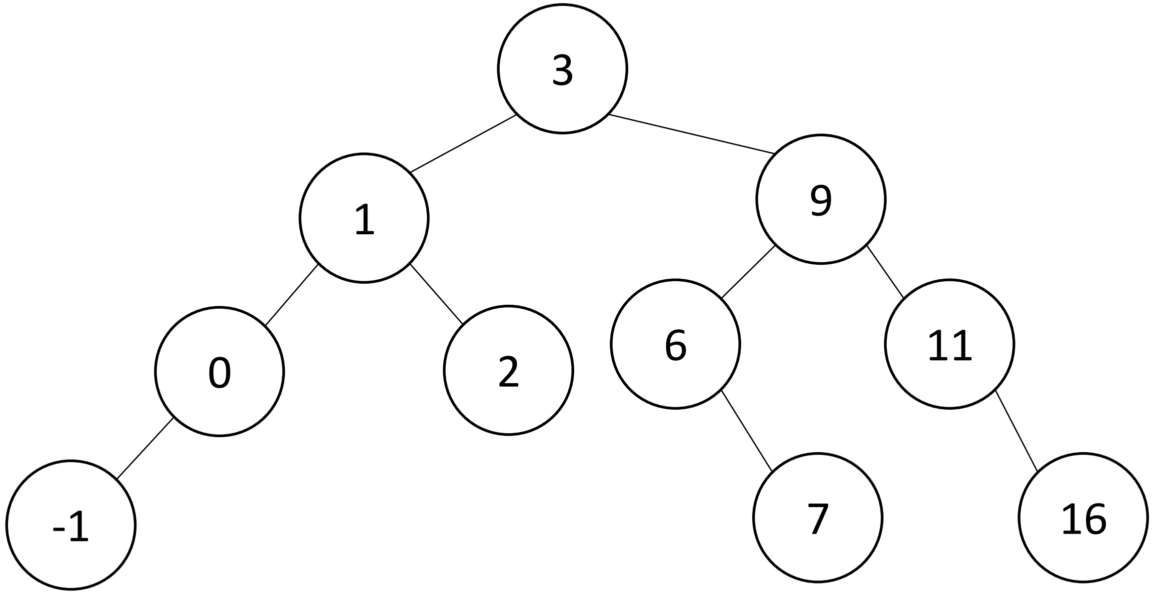


Not Balanced!



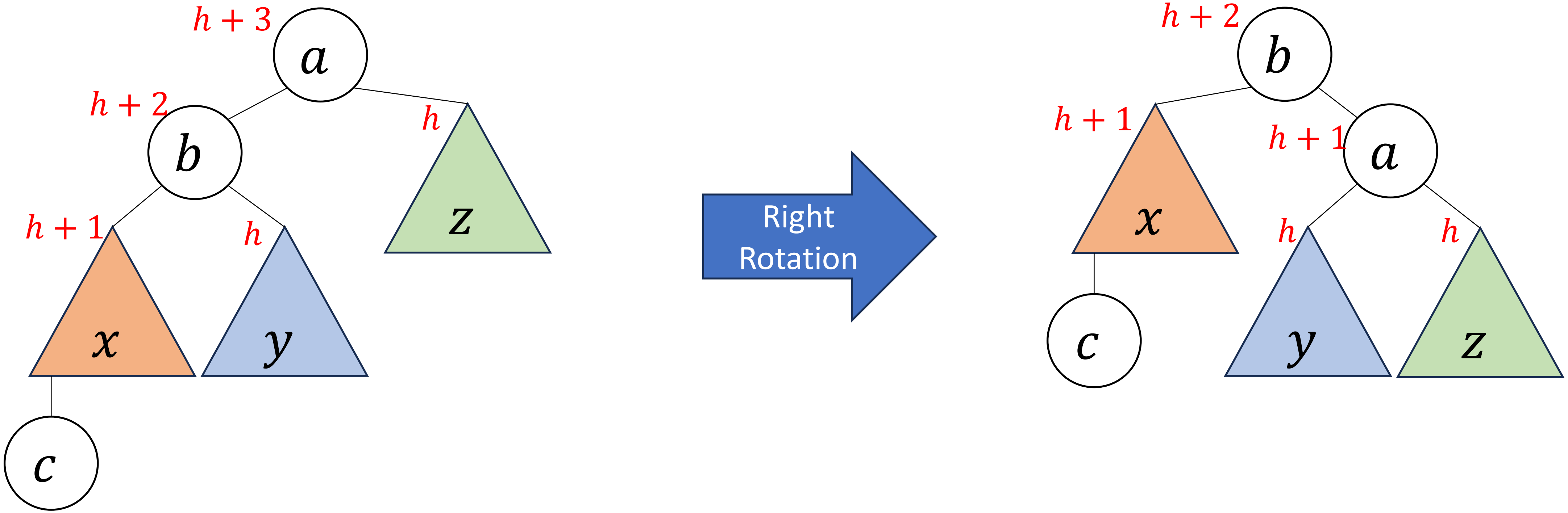

3
1
9
0
2
6
11
-1
7
16

Balanced!

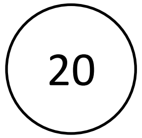


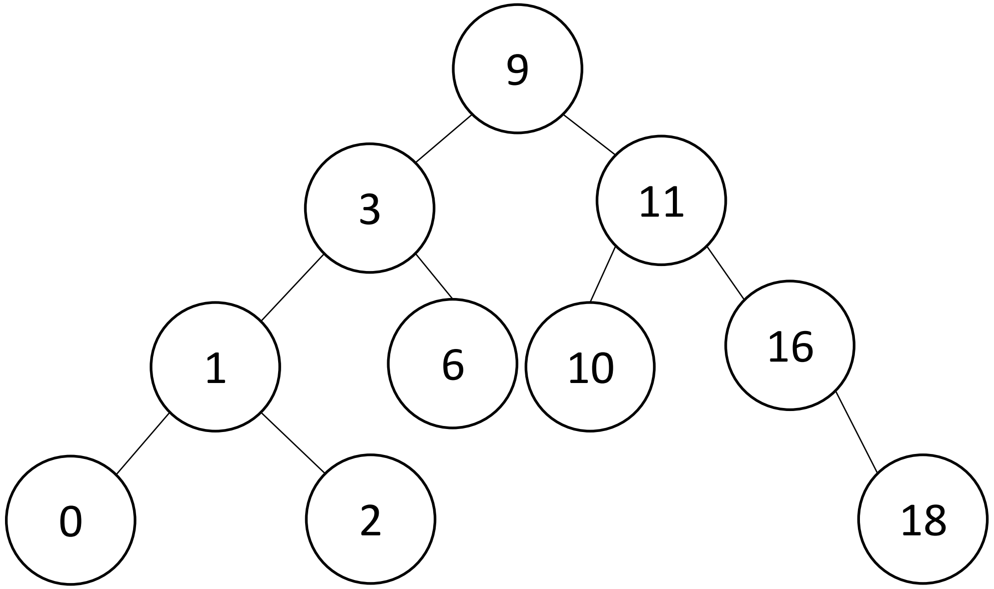
Right Rotation

- Make the left child the new root
- Make the old root the right child of the new
- Make the new root's right subtree the old root's left subtree

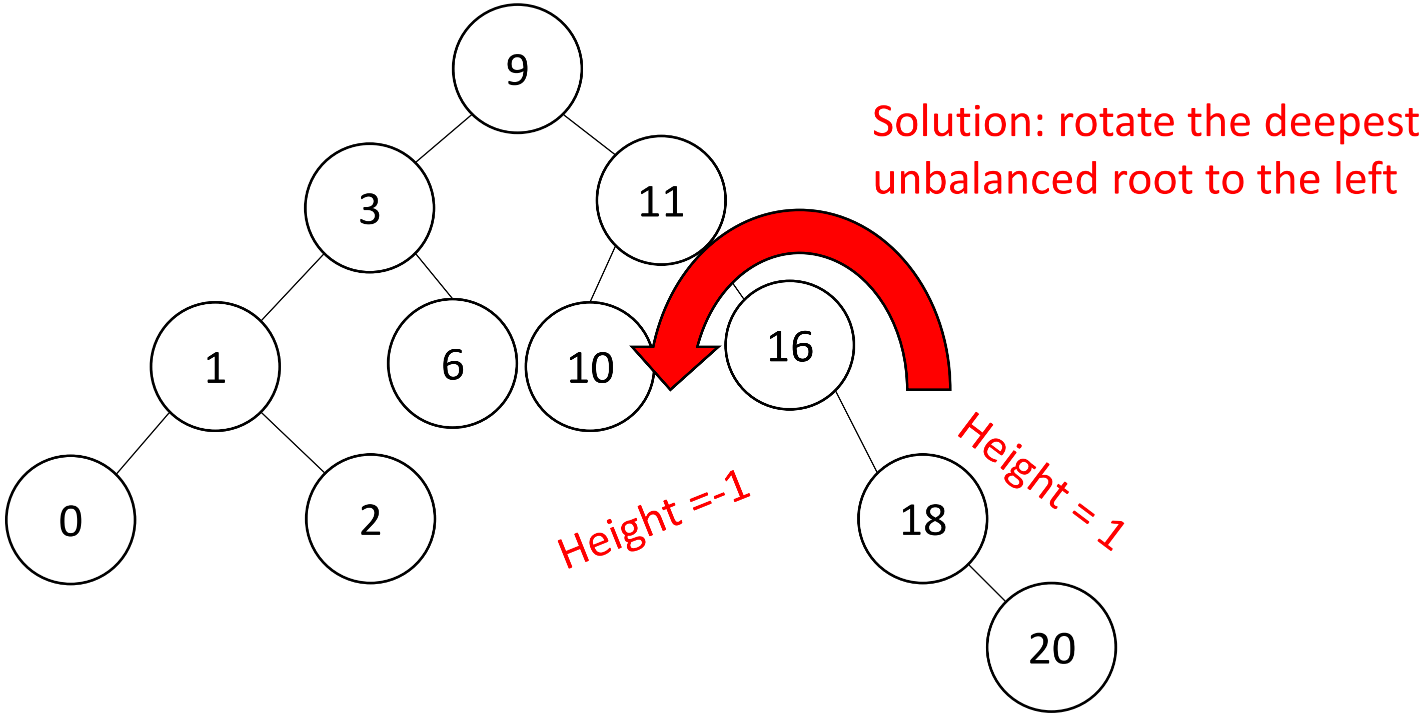


Insert Example

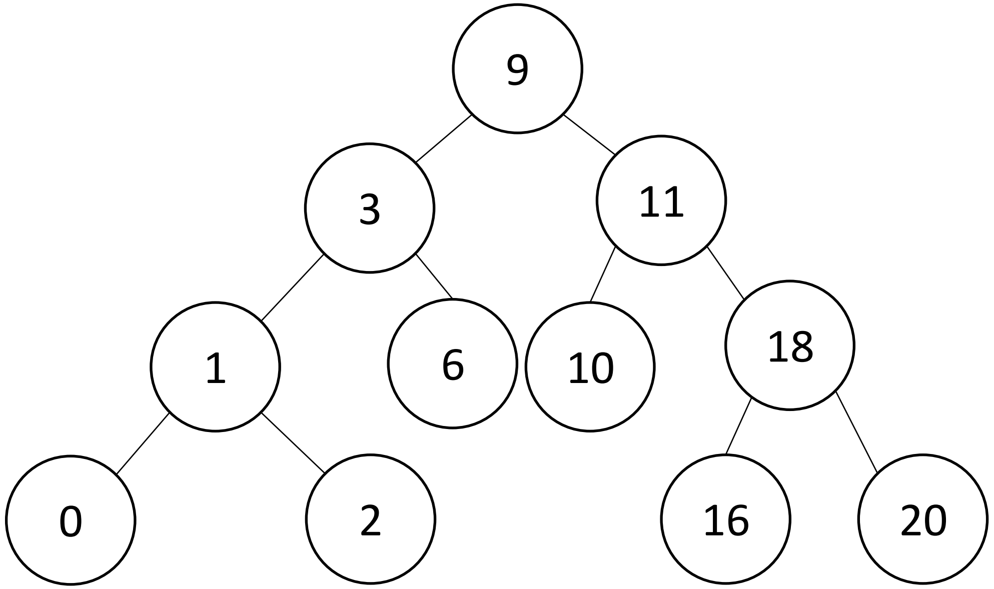




Not Balanced!

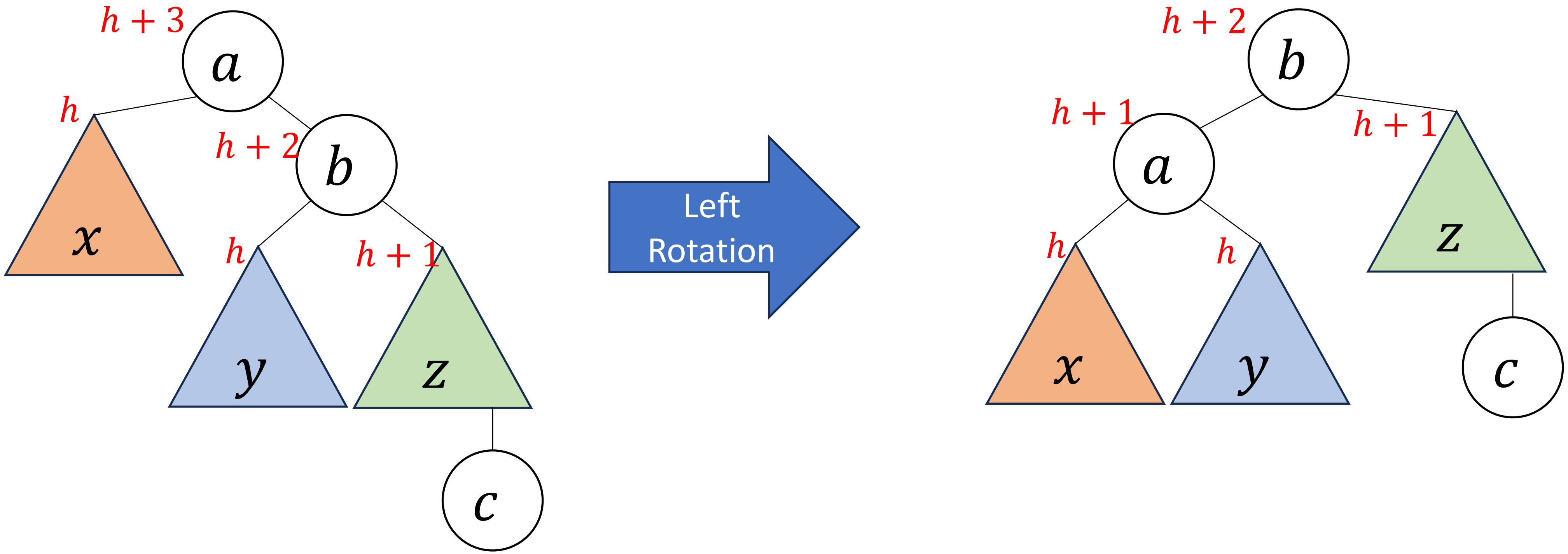


Balanced!



Left Rotation

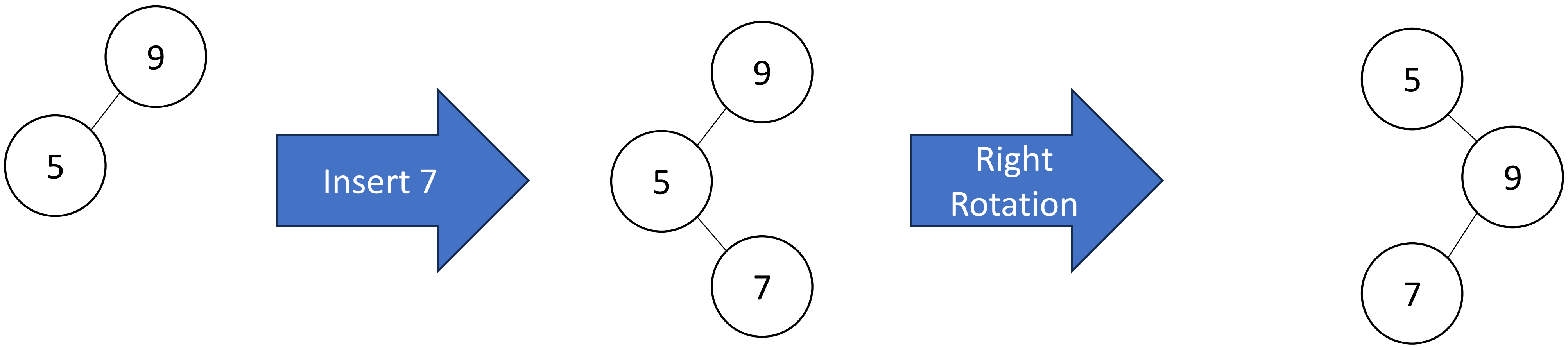
- Make the right child the new root
- Make the old root the left child of the new
- Make the new root's left subtree the old root's right subtree



Insertion Story So Far

- After insertion, update the heights of the node's ancestors
- Check for unbalance
- If unbalanced then at the deepest unbalanced root:
  - If the left subtree was deeper then rotate right
  - If the right subtree was deeper then rotate left

This is incomplete!
There are some cases
where this doesn't work!



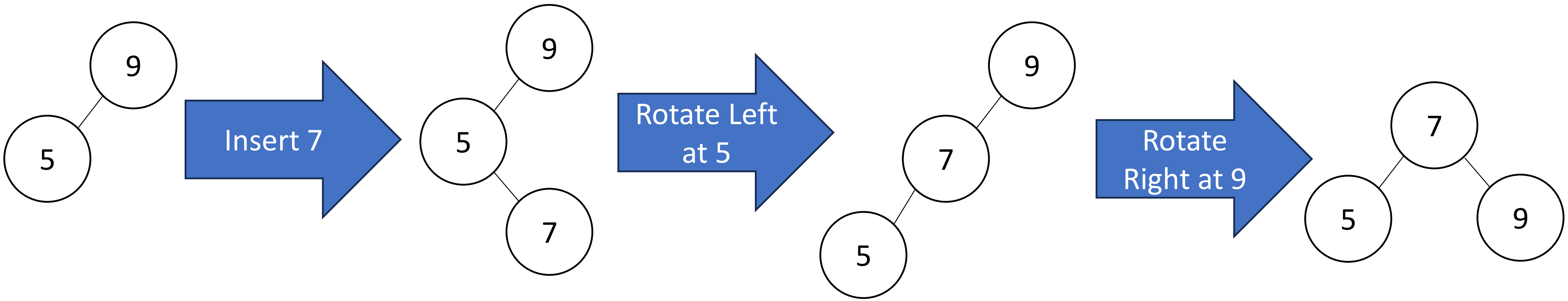
Insertion Story So Far

- After insertion, update the heights of the node's ancestors
- Check for unbalance
- If unbalanced then at the deepest unbalanced root:
  - Case LL: If we inserted in the **left** subtree of the **left** child then rotate right
  - Case RR: If we inserted in the **right** subtree of the **right** child then rotate left
  - Case LR: If we inserted into the **right** subtree of the **left** child then ???
  - Case RL: If we inserted into the **left** subtree of the **right** child then ???

Cases LR and RL require 2 rotations!

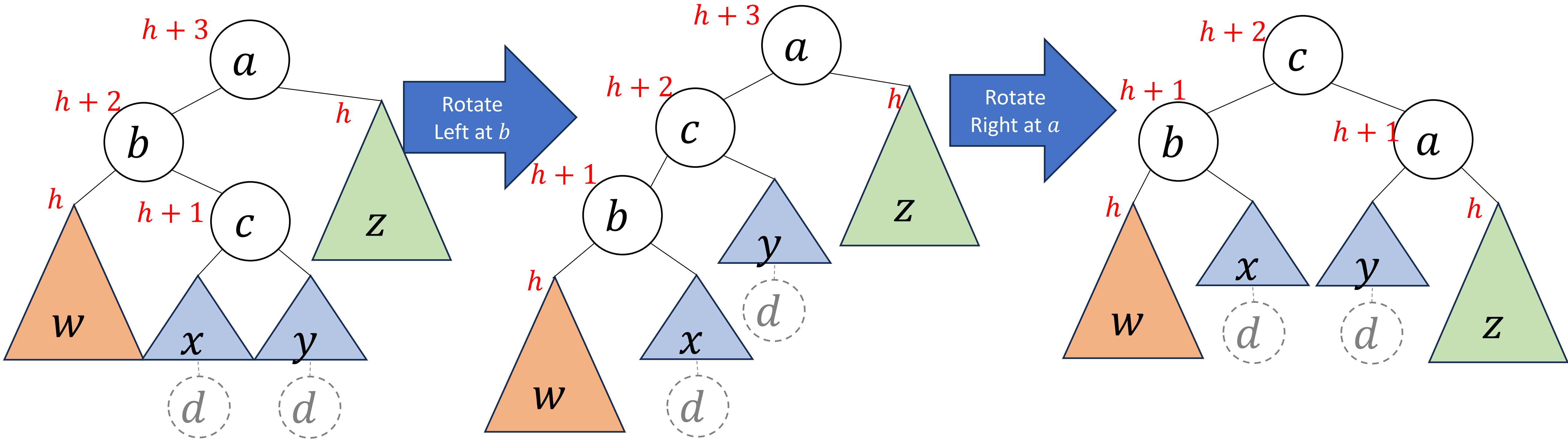
Case LR

- From deepest unbalanced root:
  - Rotate left at the left child
  - Rotate right at the root



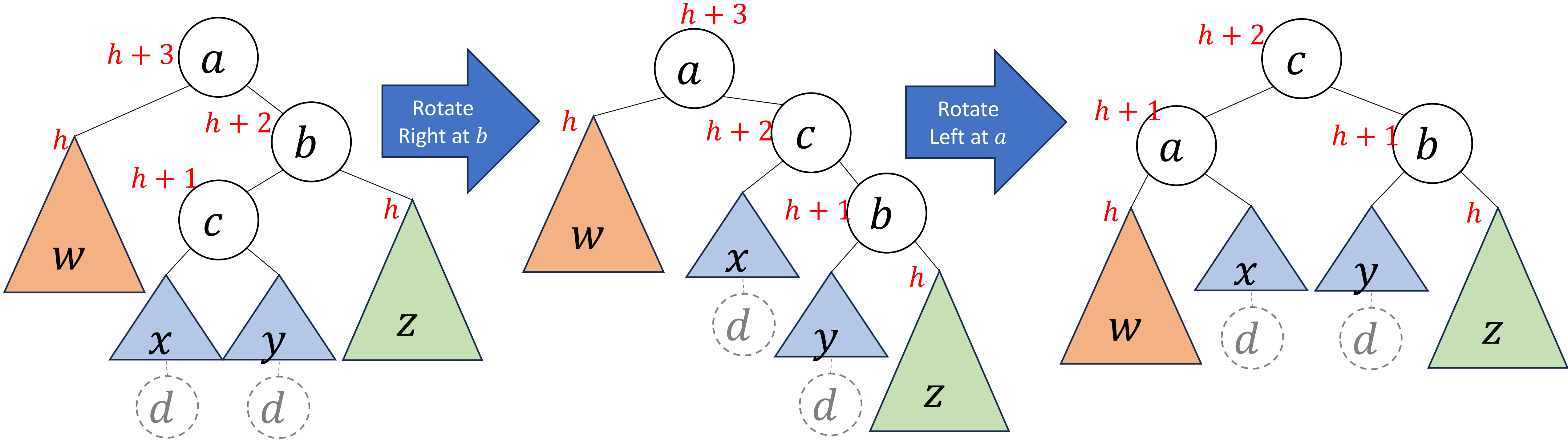
Case LR in General

- Imbalance caused by inserting in the left child's right subtree
- Rotate left at the left child
- Rotate right at the unbalanced node



Case RL in General

- Imbalance caused by inserting in the right child's left subtree
- Rotate right at the right child
- Rotate left at the unbalanced node



Insert Summary

- After a BST insertion, update the heights of the node's ancestors
- From leaf to root, check if each node is unbalanced
- If a node is unbalanced then at the deepest unbalanced node:
  - Case LL: If we inserted in the **left** subtree of the **left** child then: rotate right
  - Case RR: If we inserted in the **right** subtree of the **right** child then: rotate left
  - Case LR: If we inserted into the **right** subtree of the **left** child then: rotate left at the left child and then rotate right at the root
  - Case RL: If we inserted into the **left** subtree of the **right** child then: rotate right at the right child and then rotate left at the root
- Done after either reaching the root or applying **one** of the above cases

Delete Summary

- Tldr: same cases, reverse direction of rotation, may need to repeat with ancestors
- After a BST deletion, update the heights of the node’s ancestors
- From leaf to root, check if each node is unbalanced
- If a node is unbalanced then at the deepest unbalanced node:
  - Case LL: If we deleted in the **left** subtree of the **left** child then: rotate left
  - Case RR: If we deleted in the **right** subtree of the **right** child then: rotate right
  - Case LR: If we deleted into the **right** subtree of the **left** child then: rotate right at the left child and then rotate left at the root
  - Case RL: If we deleted into the **left** subtree of the **right** child then: rotate left at the right child and then rotate right at the root
- Continue checking until reach the root

Other Tree-based Dictionaries

- Red-Black Trees
  - Similar to AVL Trees in that we add shape rules to BSTs
  - More “relaxed” shape than an AVL Tree
    - Trees can be taller (though not asymptotically so)
    - Needs to move nodes less frequently
  - This is what Java’s TreeMap uses!
- Tries
  - Similar to a Huffman Tree
  - Requires keys to be sequences (e.g. Strings)
  - Combines shared prefixes among keys to save space
  - Often used for text-based searches
    - Web search
    - Genomes