COMP90038
Algorithms and Complexity

Lecture 15: Balanced Trees

(with thanks to Harald Søndergaard & Michael Kirley)

Casey Myers
Casey.Myers@unimelb.edu.au
David Caro Building (Physics) 274

Review from Lecture 14

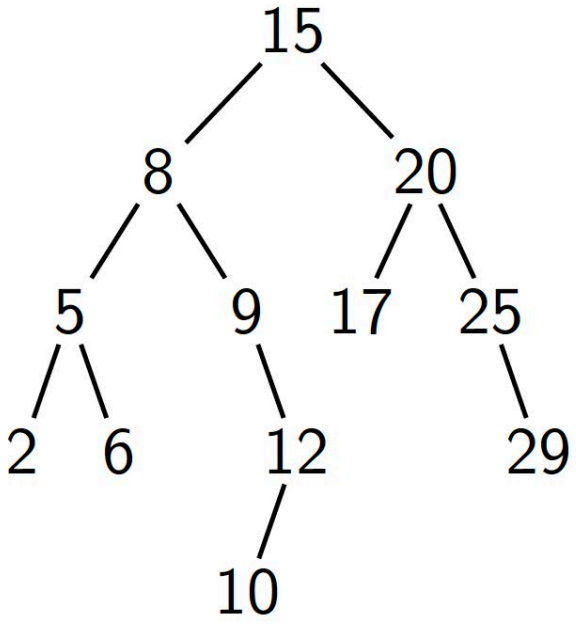
- A binary search tree, or BST, is a binary tree that stores elements in all internal nodes, with each sin-tree satisfying the BST property:

  Let the root be r; then each element in the left subtree is smaller that r and each element in the right sub-tree is larger than r. (For simplicity, we assume that all keys are different.)

Review from Lecture 14

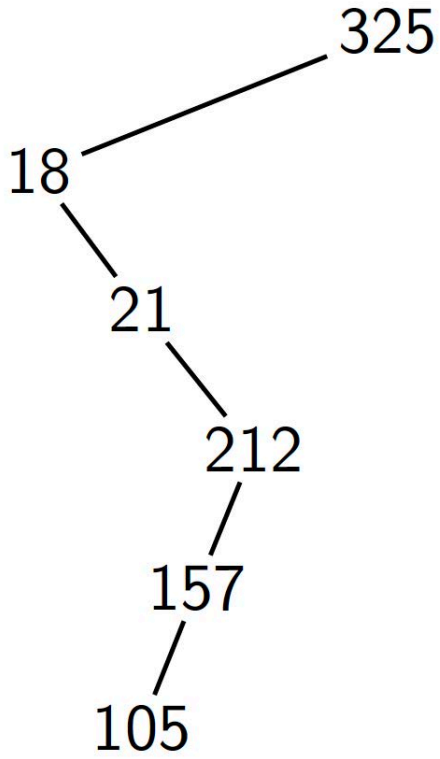
- BSTs are useful for search applications. To search for k in a BST, compare against its root r. If $r = k$, we are done; otherwise search in the left or right sub-tree, according to $k < r$ or $k > r$.

- If a BST with n elements is "reasonably" balanced, search involves in the worst case, $\Theta(\log n)$ comparisons.



Review from Lecture 14

- If the BST is not well balanced, search performance degrades, and may be as bad as linear search:



Approaches to Balanced Binary Search Trees

- To optimise the performance of BST search, it is important to keep trees (reasonably) balanced
- Instance simplification approaches: Self-balancing trees:
  - AVL trees
  - Red-black trees
  - Splay trees
- Representational changes:
  - 2-3 trees
  - 2-3-4 trees
  - B-trees

Approaches to Balanced Binary Search Trees

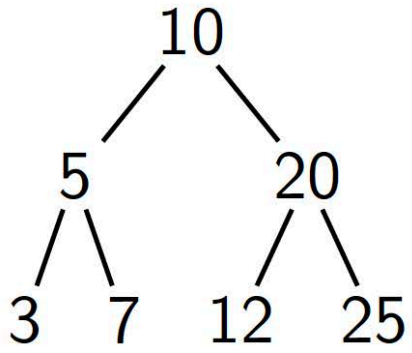
- To optimise the performance of BST search, it is important to keep trees (reasonably) balanced
- Instance simplification approaches: Self-balancing trees:
  - AVL trees
  - Red-black trees
  - Splay trees
- Representational changes:
  - 2-3 trees
  - 2-3-4 trees
  - B-trees

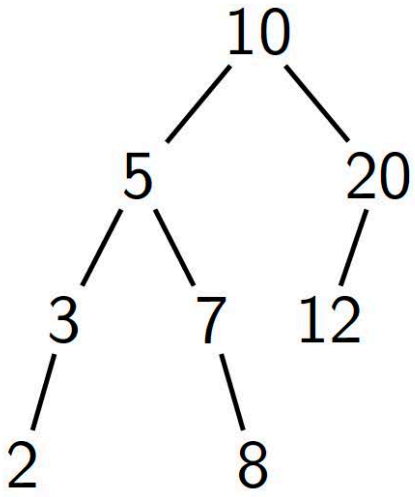
AVL Trees

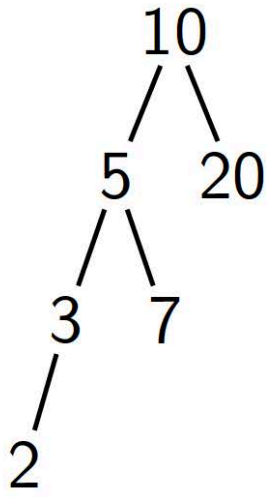
- Named after Adelson-Velsky and Landis
- Recall that we defined the height of the empty tree as -1.
- For a binary (sub-) tree, let the balance factor be the difference between the height of its left sub-tree and that of its right sub-tree.
- An AVL tree is a BST in which the balance factor is -1, 0, or 1, for every sub-tree.

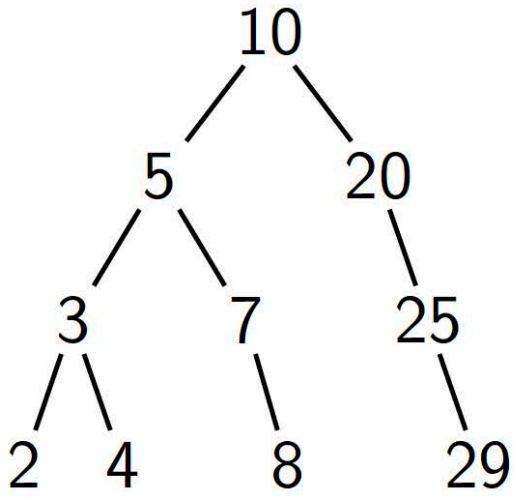
AVL Trees: Examples and Counter-Examples

- Which of these are AVL trees?



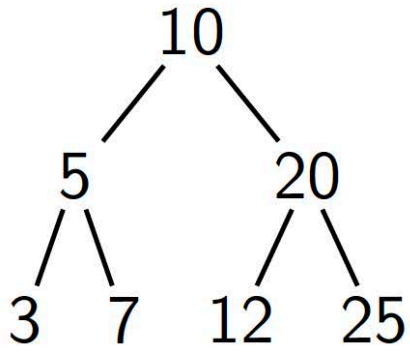


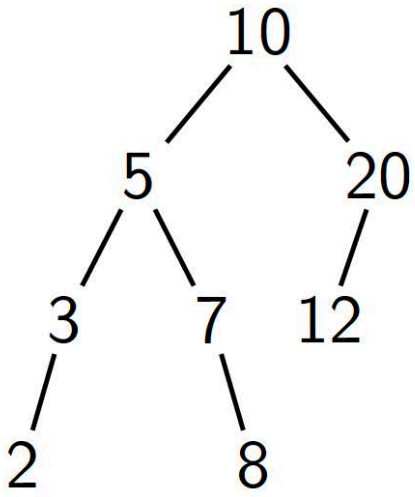


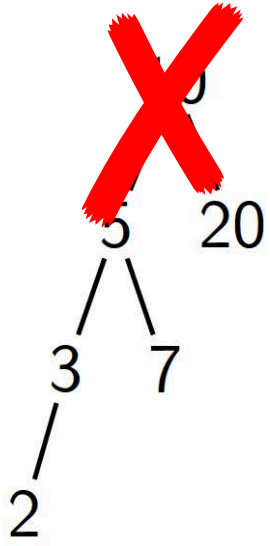


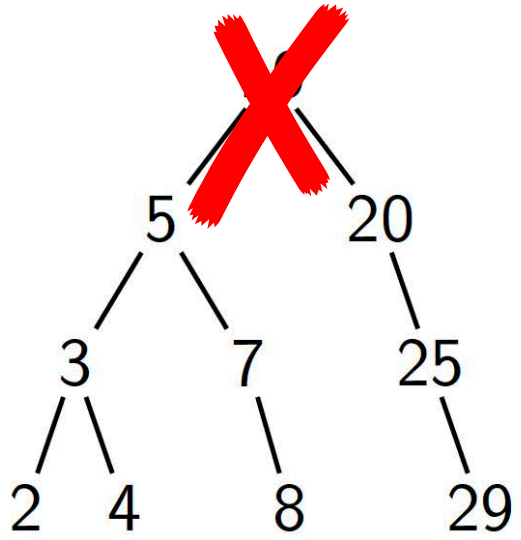
AVL Trees: Examples and Counter-Examples

- Which of these are AVL trees?



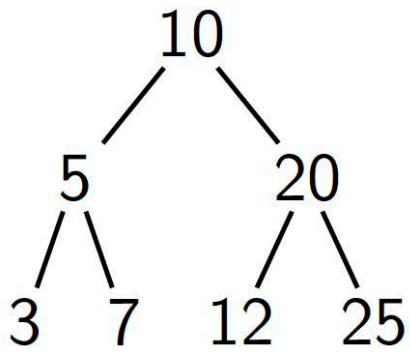


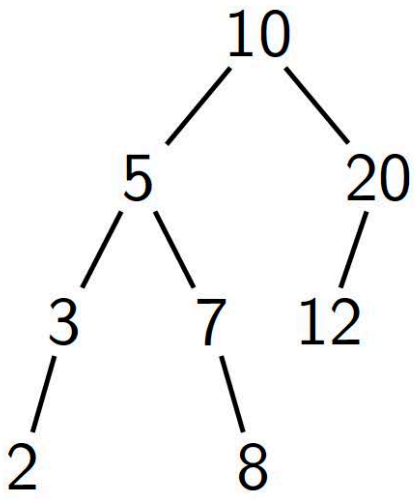




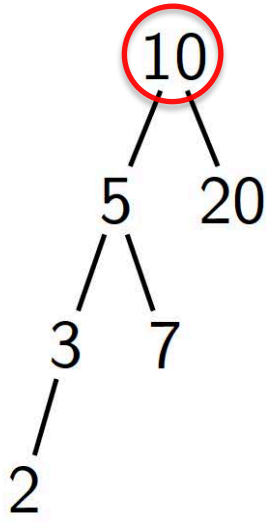
AVL Trees: Examples and Counter-Examples

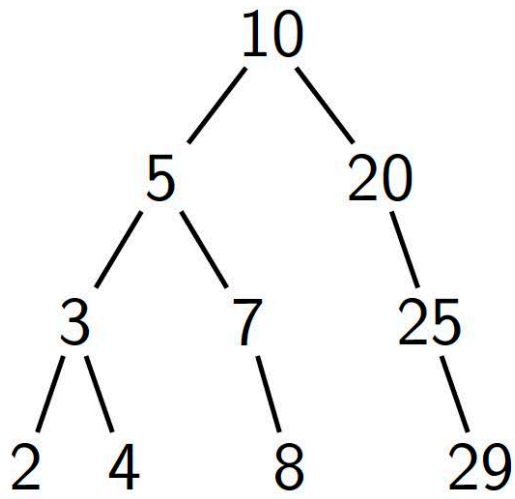
- Which of these are AVL trees?





H: 3, 1; BF: 3-1 = 2

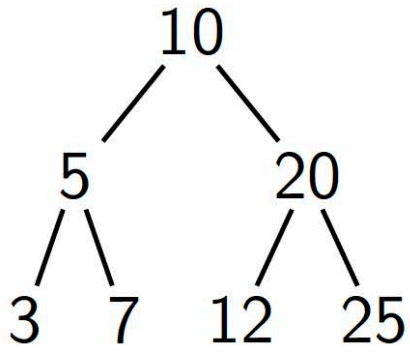


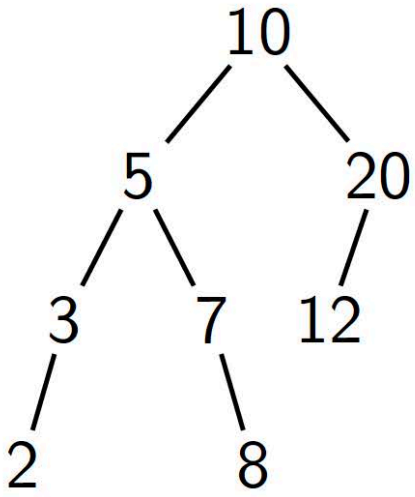


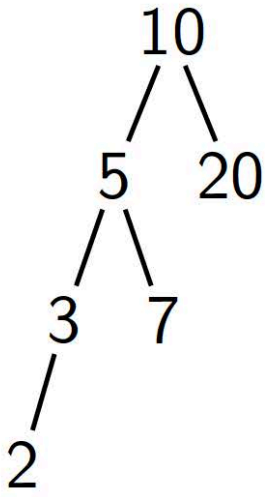
AVL Trees: Examples and Counter-Examples

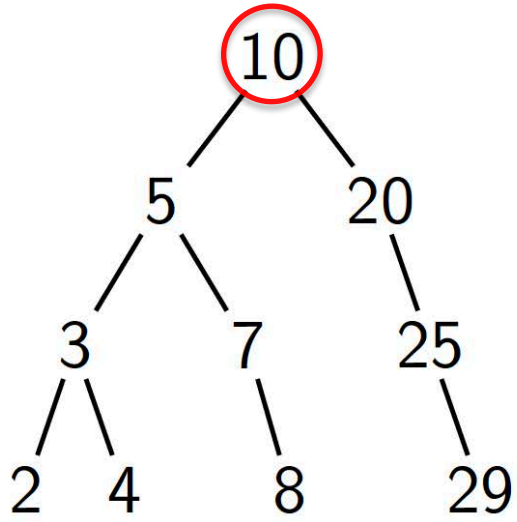
- Which of these are AVL trees?

H: 3, 3; BF: 3-3 = 0





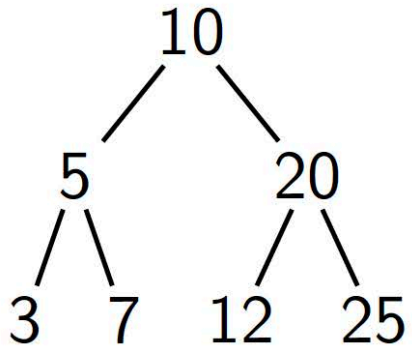


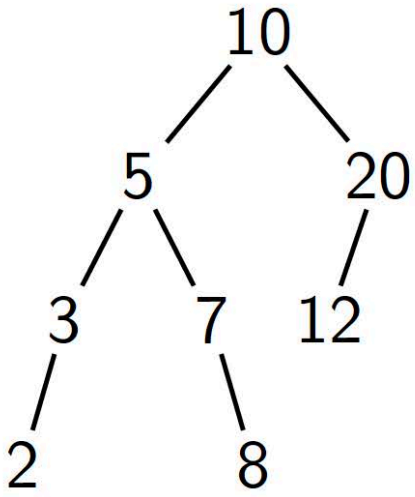


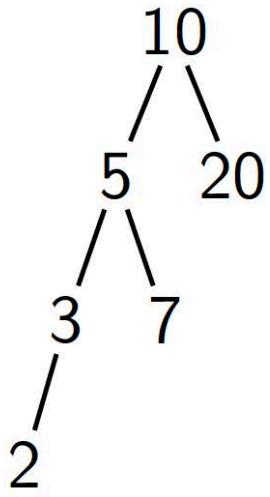
AVL Trees: Examples and Counter-Examples

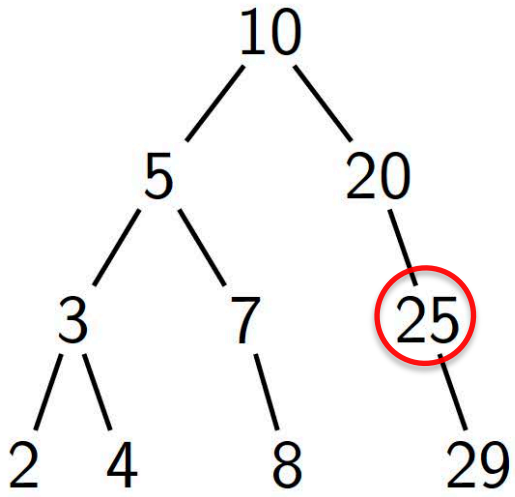
- Which of these are AVL trees?

H: 0, 1; BF: 0-1 = -1





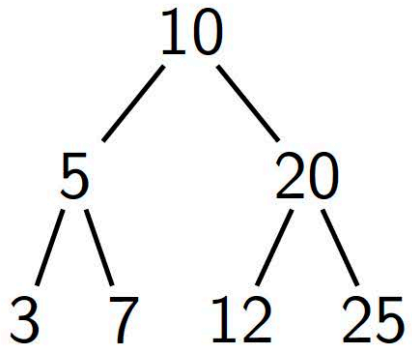


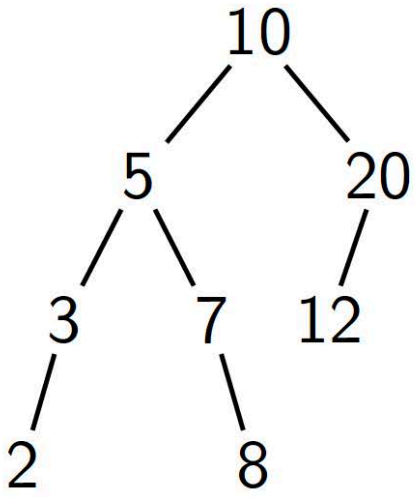


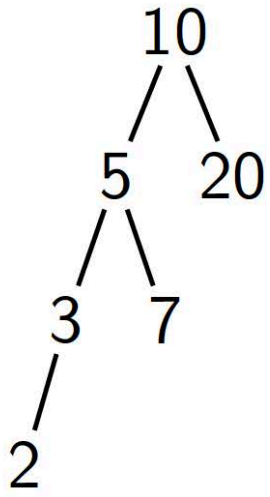
AVL Trees: Examples and Counter-Examples

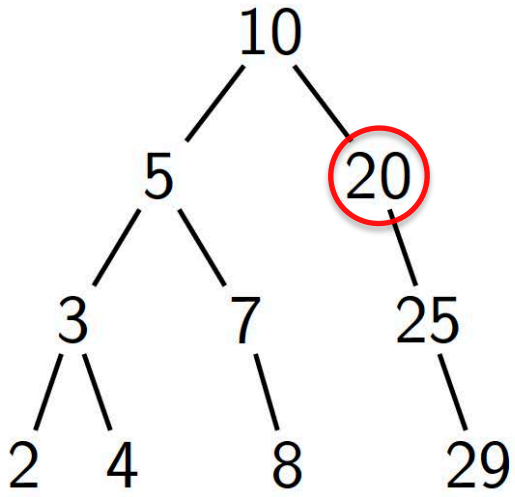
- Which of these are AVL trees?

H: 0, 2; BF: 0-2 = -2





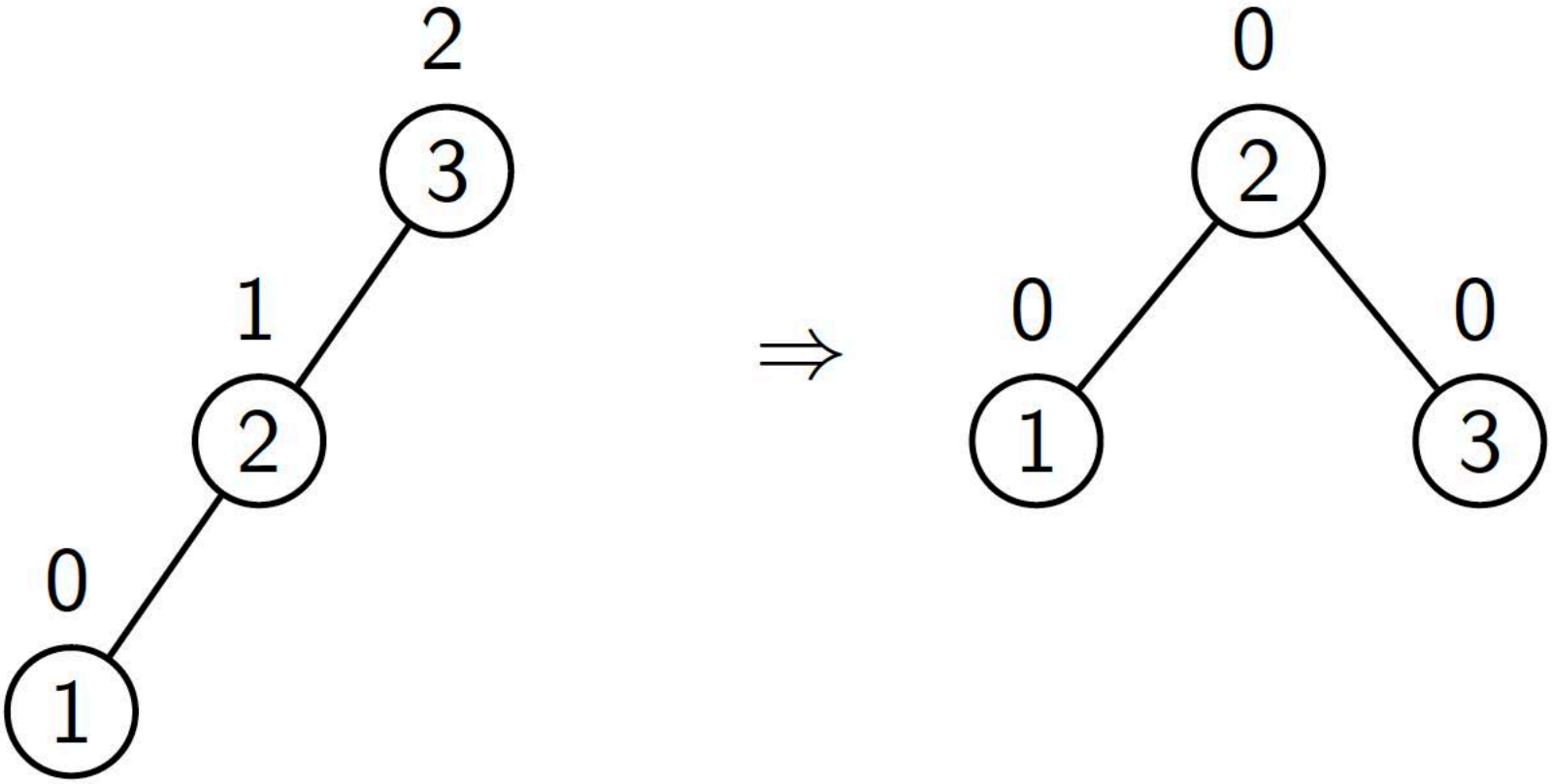




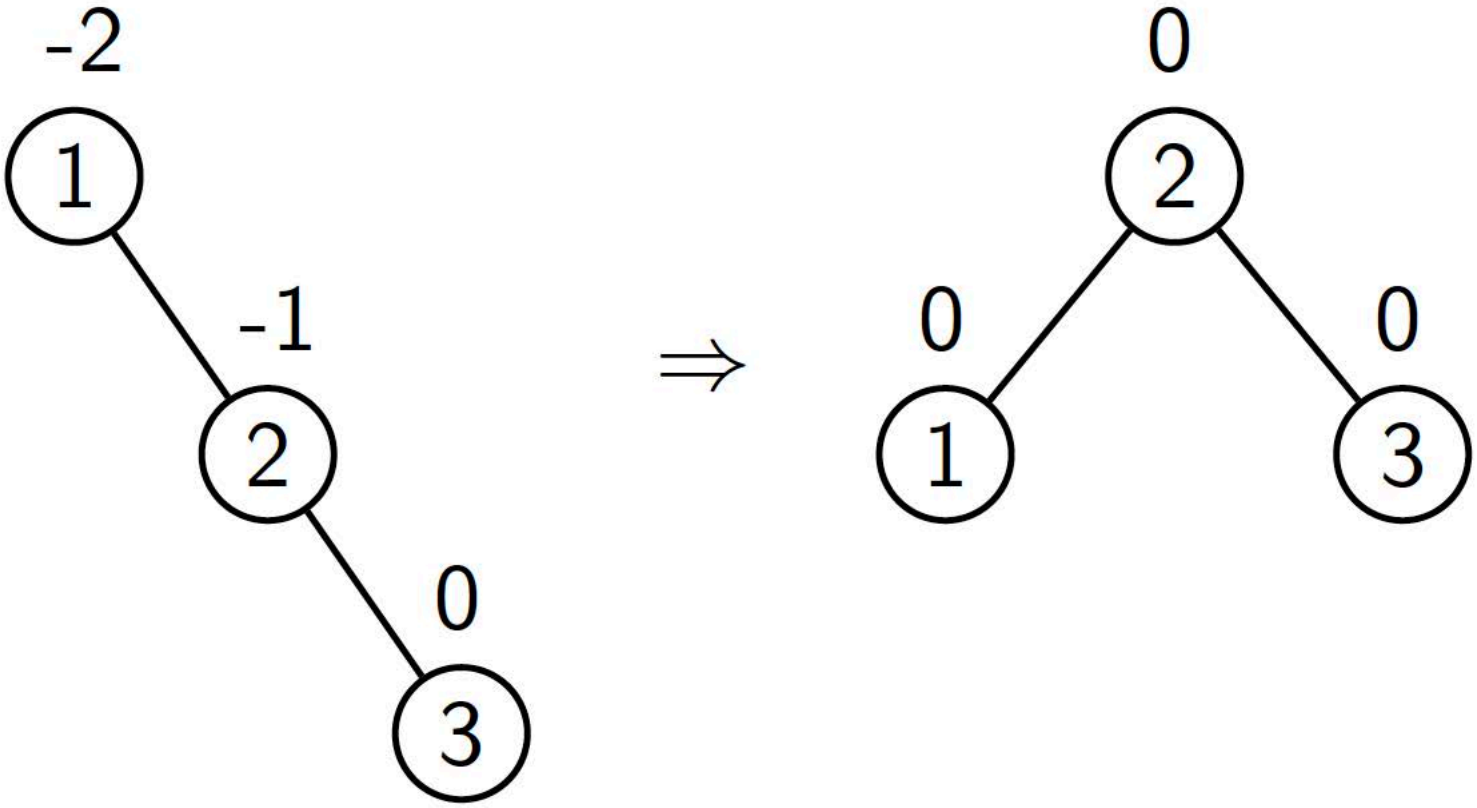
Building an AVL Tree

- As with standard BSTs, insertion of a new mode always takes place at the fringe of the tree.
- If insertion of the new node makes the AVL tree unbalanced (some nodes get balance factors of 2 or -2), transform the tree to regain its balance.
- Regaining balance can achieved with one or two simple, local transformtions, so-called rotations.

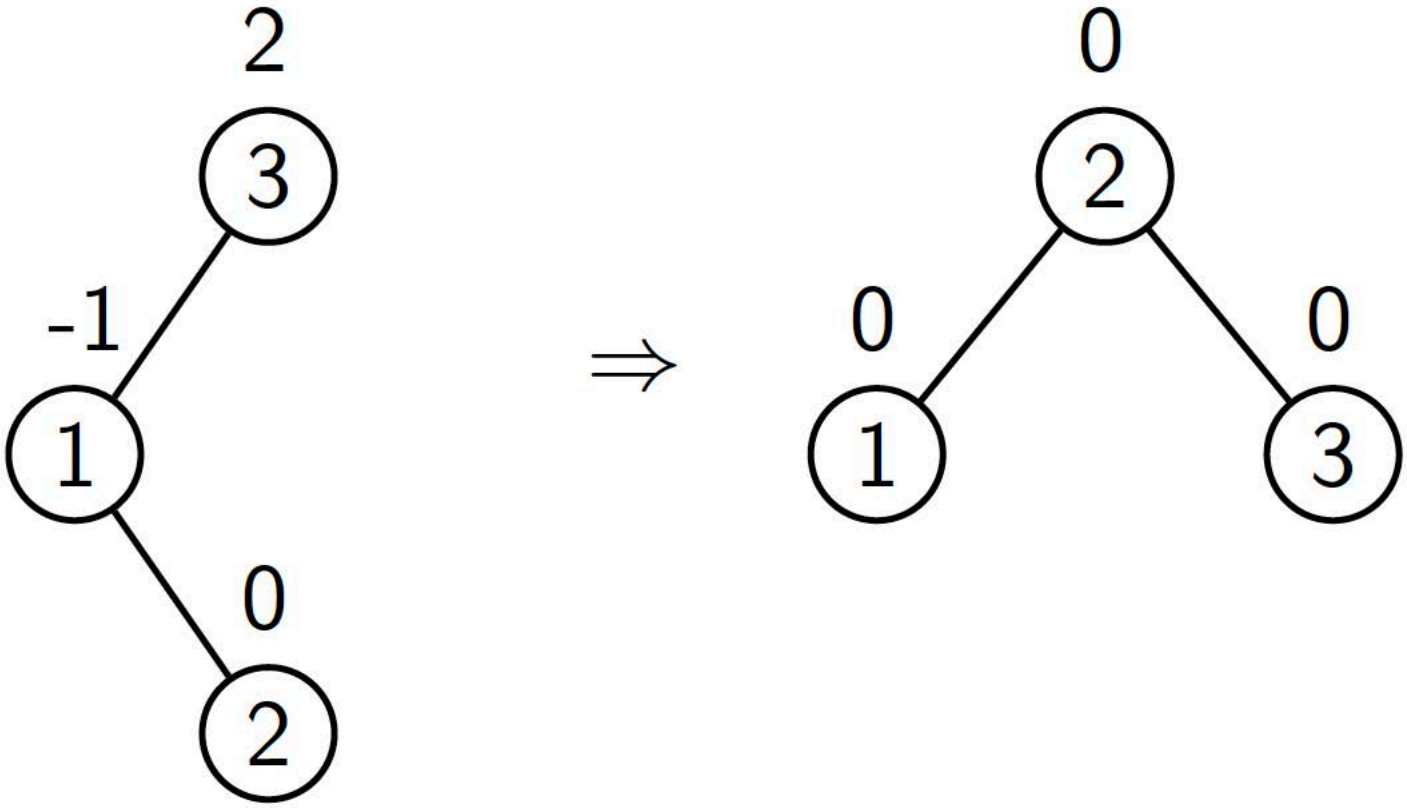
AVL Trees: R-Rotation



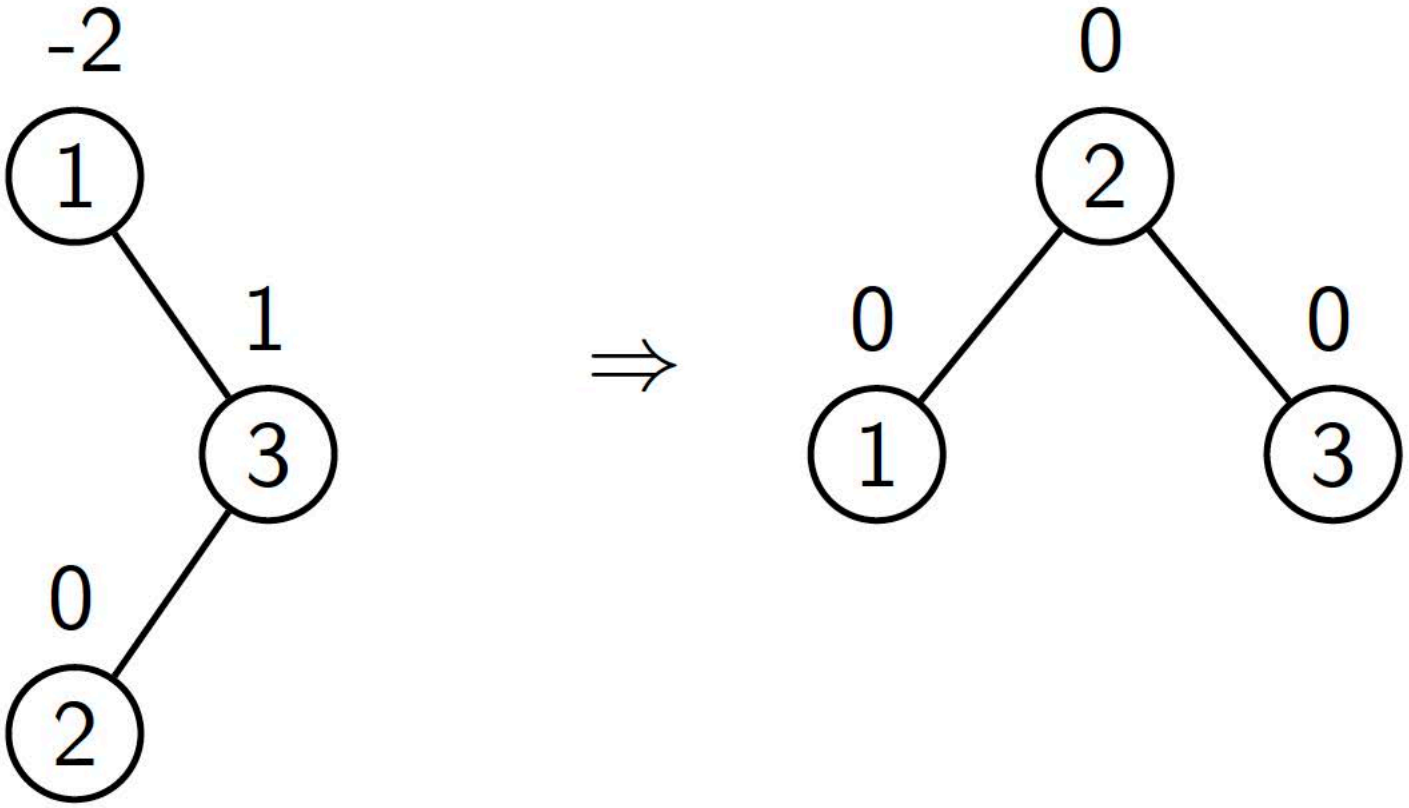
AVL Trees: L-Rotation



AVL Trees: LR-Rotation

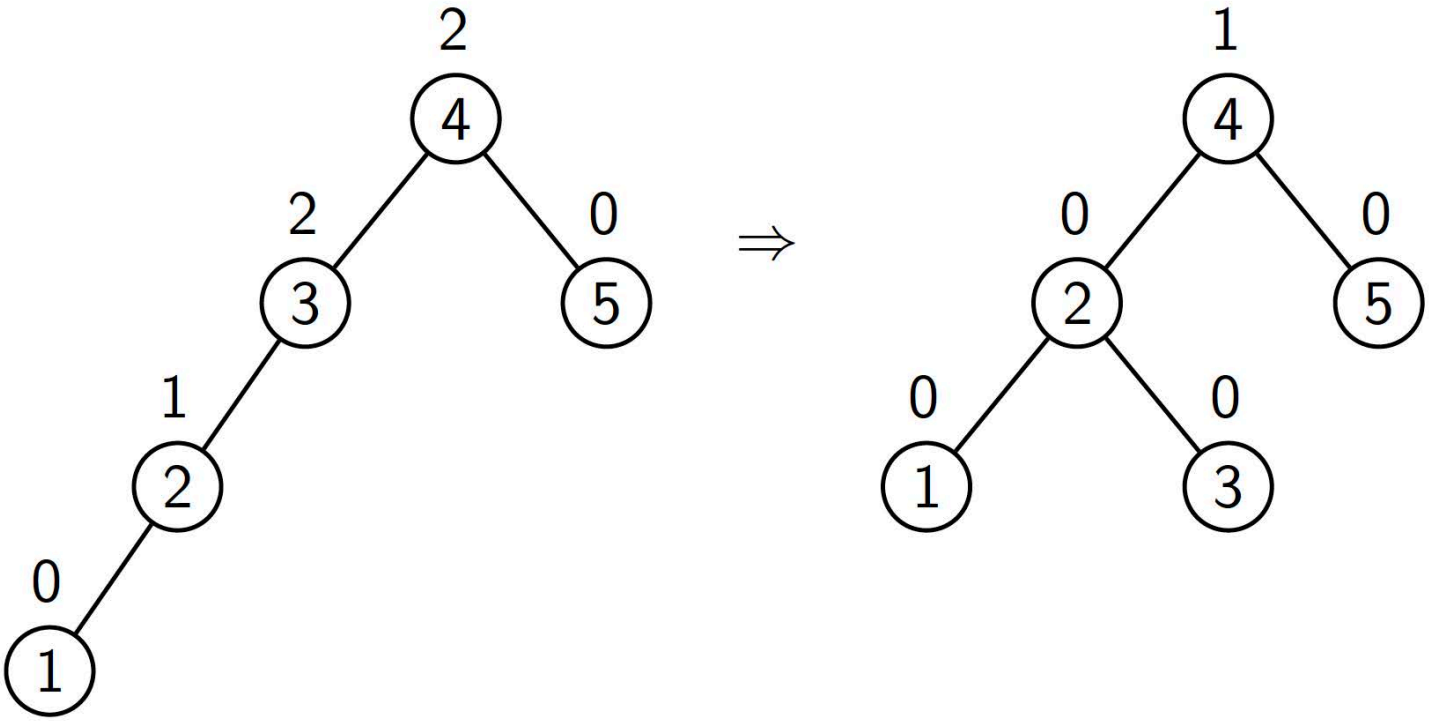


AVL Trees: RL-Rotation



AVL Trees: Where to Perform the Rotation

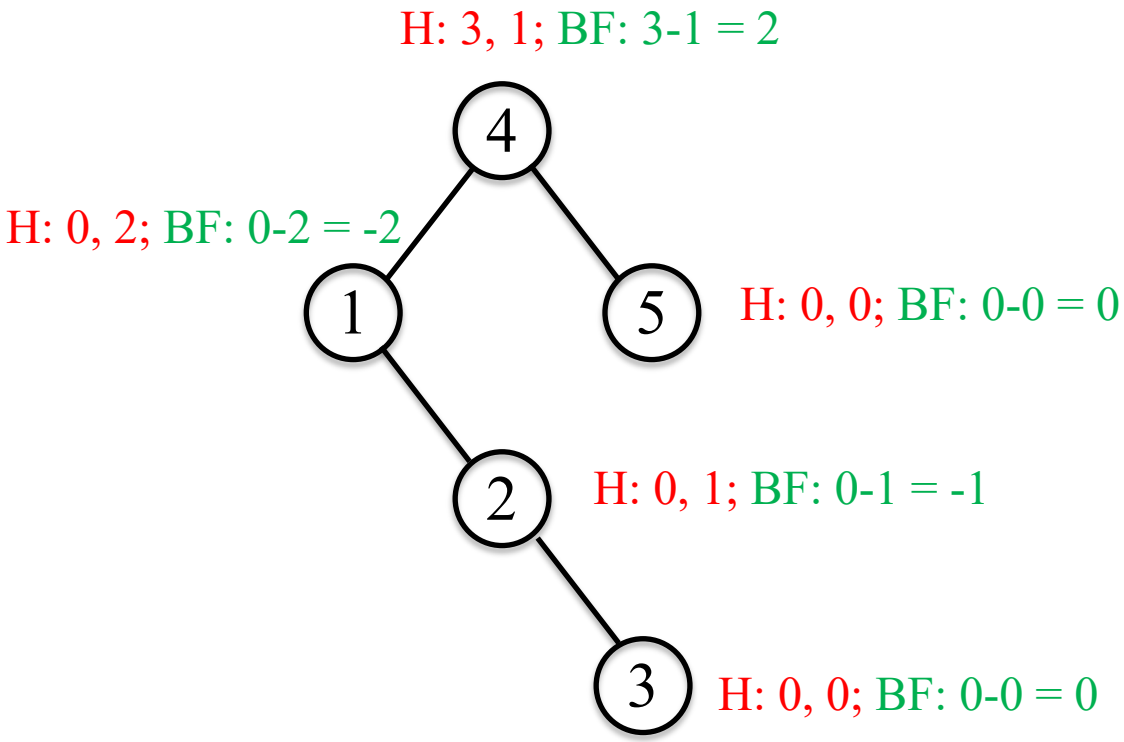
- Along an unbalanced path, we may have several nodes with balance factor 2 (or -2).



- It is always the lowest unbalanced subtree that is re-balanced.

AVL Trees: Where to Perform the Rotation

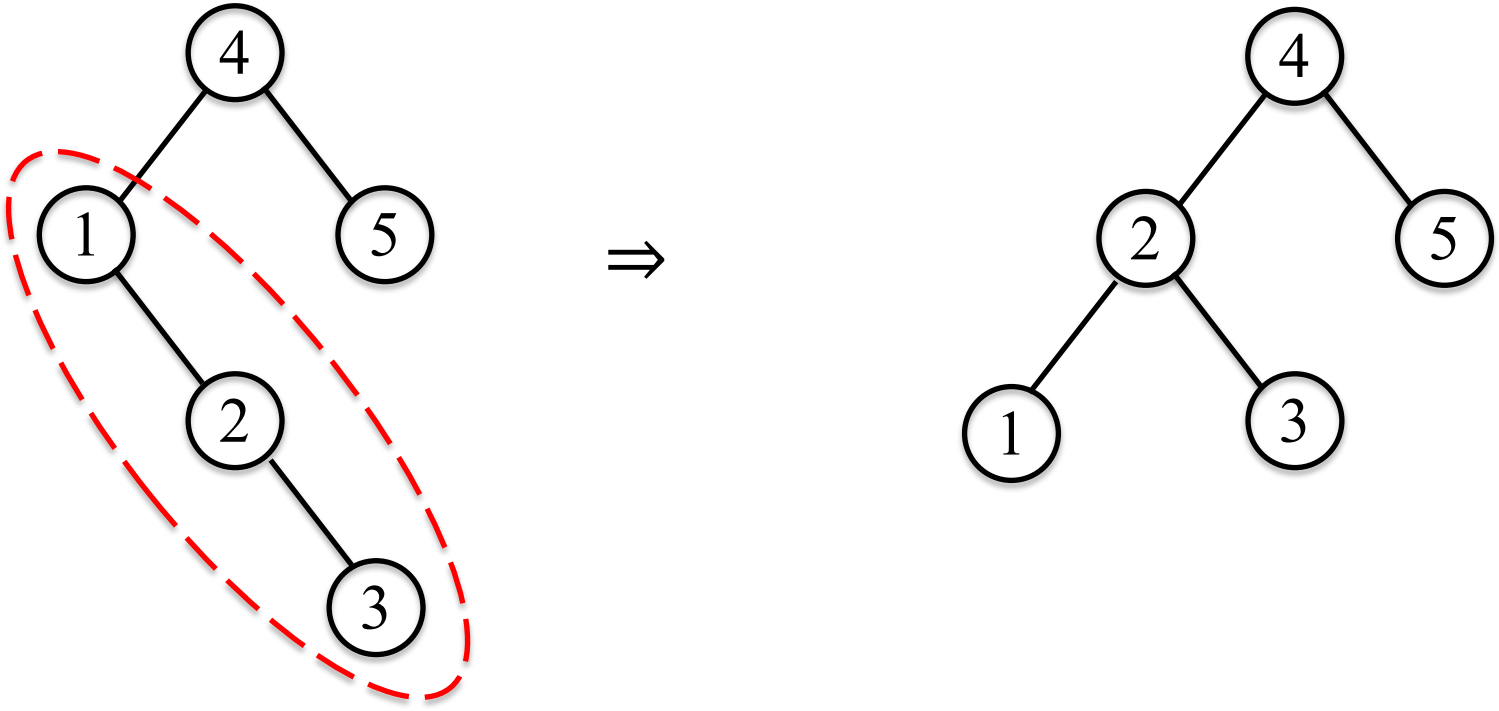
- Along an unbalanced path, we may have several nodes with balance factor 2 (or -2).



- It is always the lowest unbalanced subtree that is re-balanced.

AVL Trees: Where to Perform the Rotation

- Along an unbalanced path, we may have several nodes with balance factor 2 (or -2).



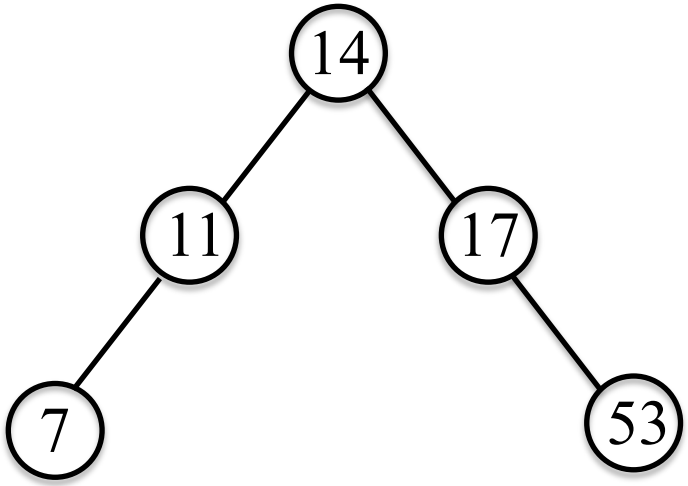
- It is always the lowest unbalanced subtree that is re-balanced.

AVL Trees: Example

- Insert each of these number into the AVL tree in order: 14, 17, 11, 7, 53, 4, 13, 12

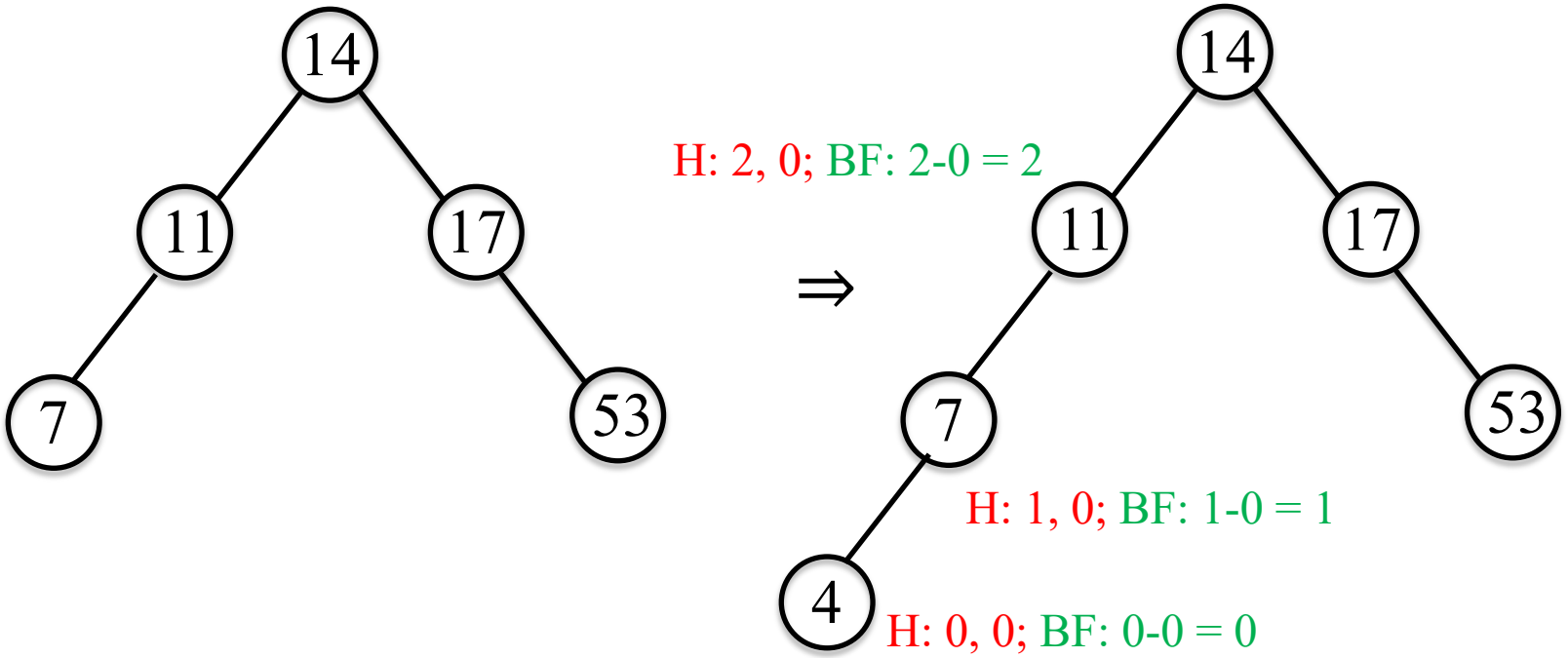
AVL Trees: Example

- Insert each of these number into the AVL tree in order: 14, 17, 11, 7, 53, 4, 13, 12



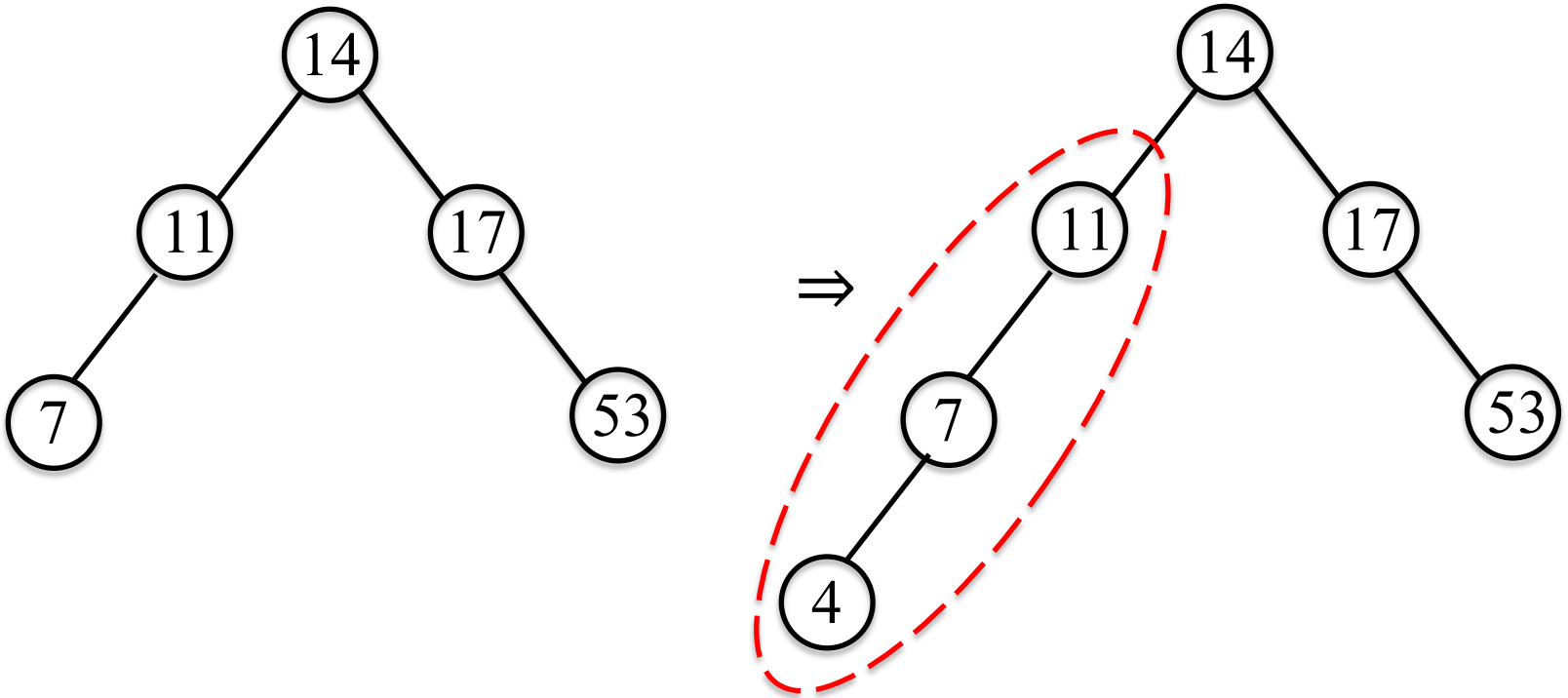
AVL Trees: Example

- Insert each of these number into the AVL tree in order: 14, 17, 11, 7, 53, 4, 13, 12



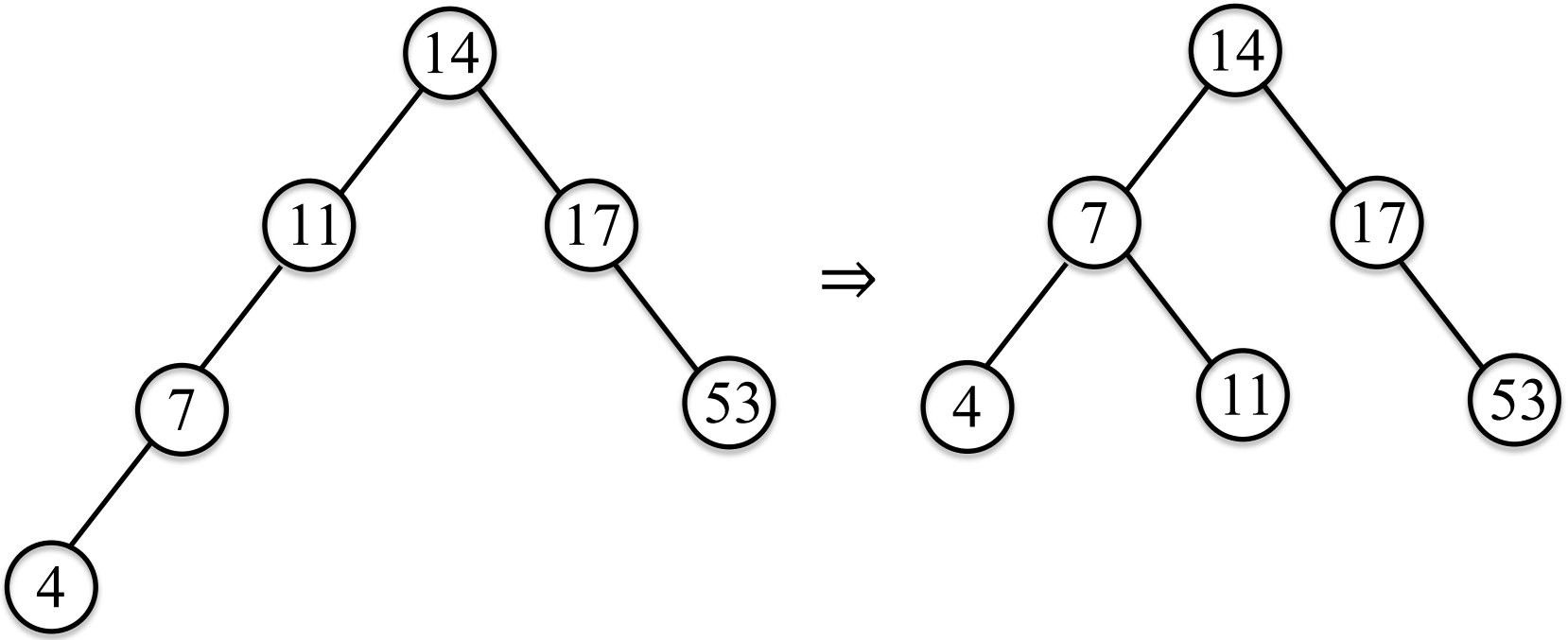
AVL Trees: Example

- Insert each of these number into the AVL tree in order: 14, 17, 11, 7, 53, 4, 13, 12



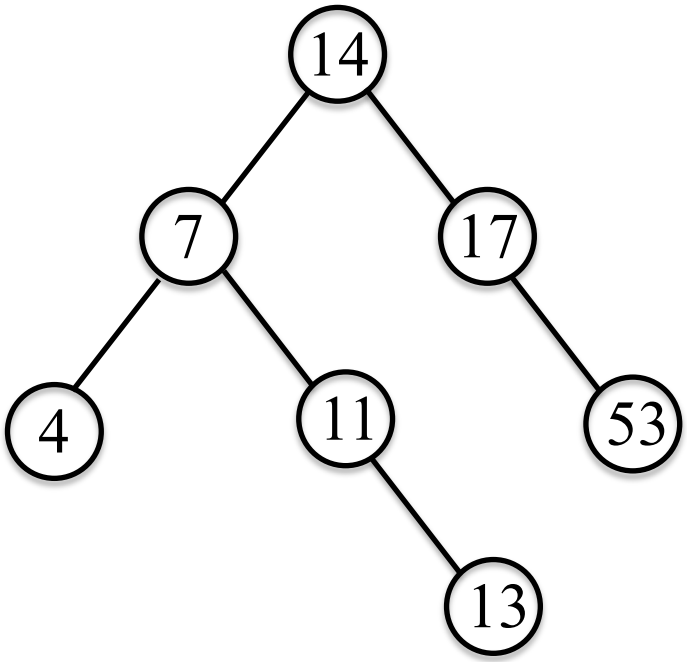
AVL Trees: Example

- Insert each of these number into the AVL tree in order: 14, 17, 11, 7, 53, 4, 13, 12



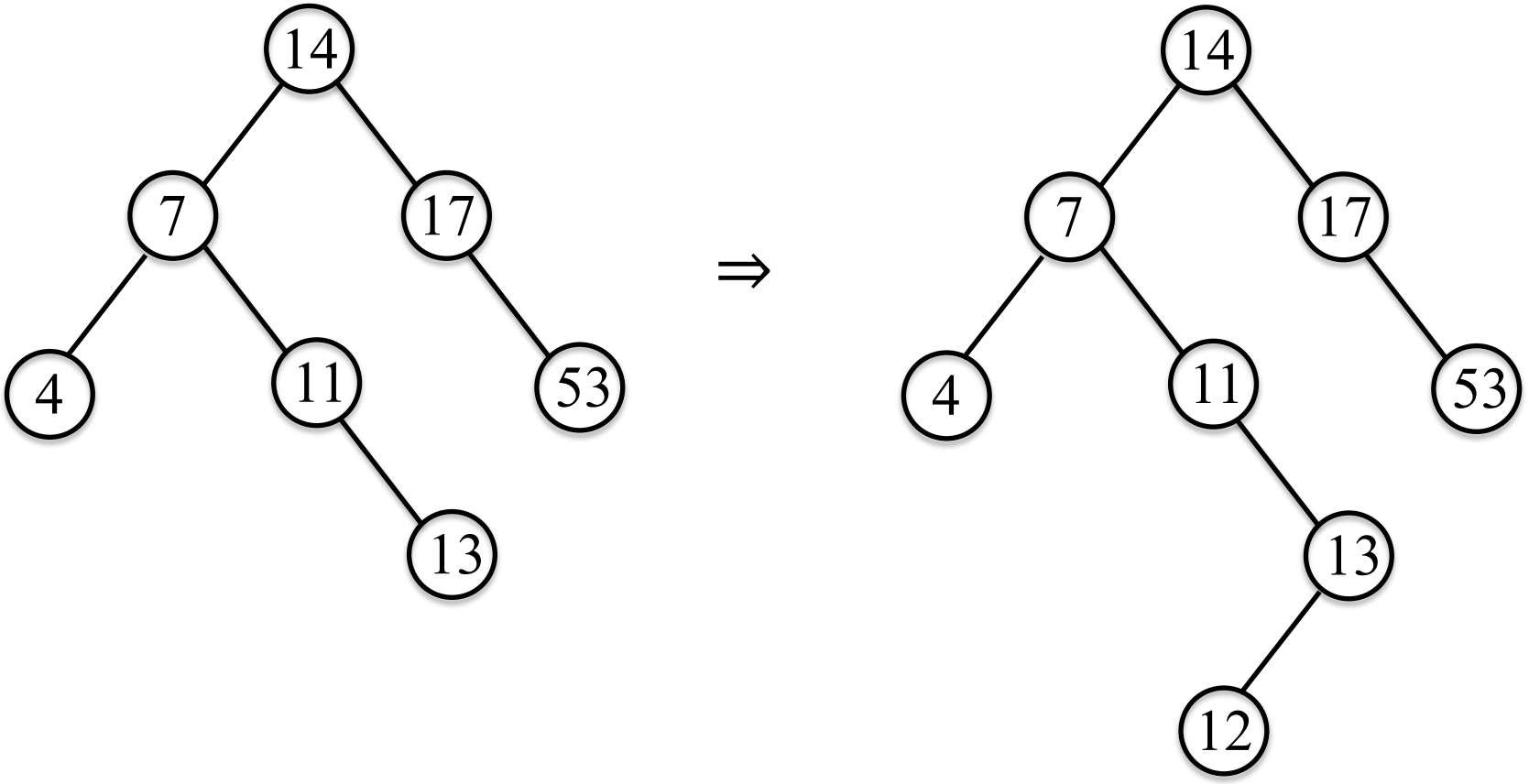
AVL Trees: Example

- Insert each of these number into the AVL tree in order: 14, 17, 11, 7, 53, 4, 13, 12



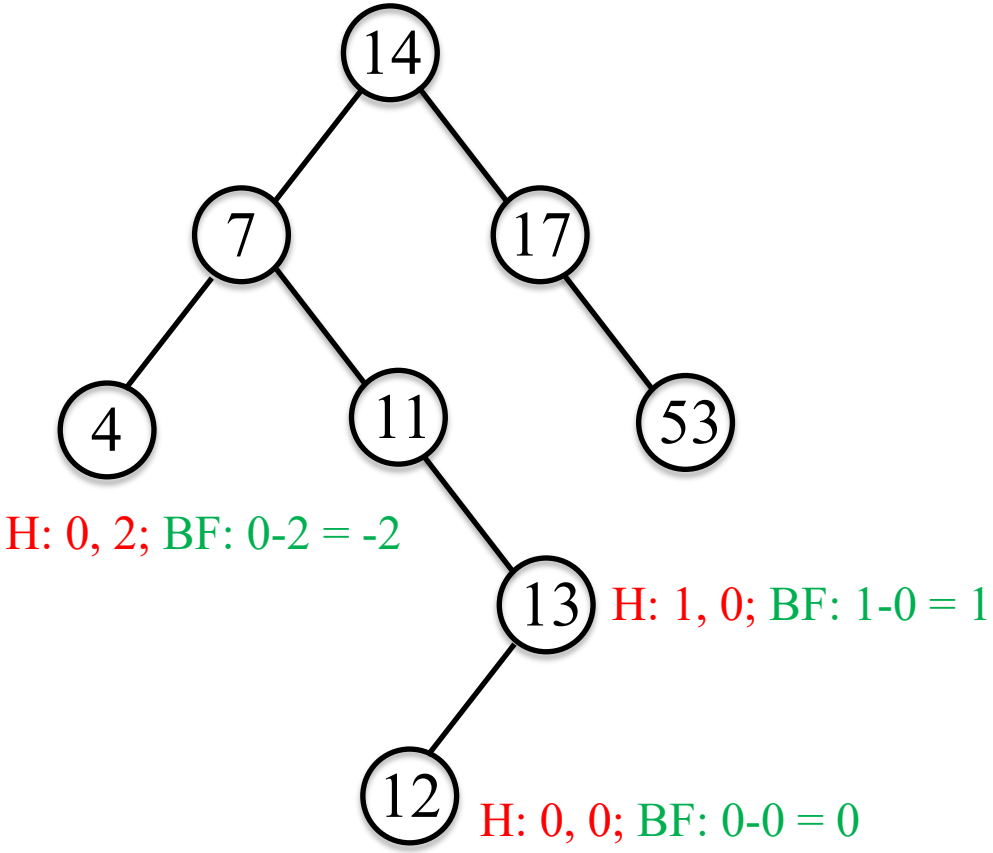
AVL Trees: Example

- Insert each of these number into the AVL tree in order: 14, 17, 11, 7, 53, 4, 13, 12



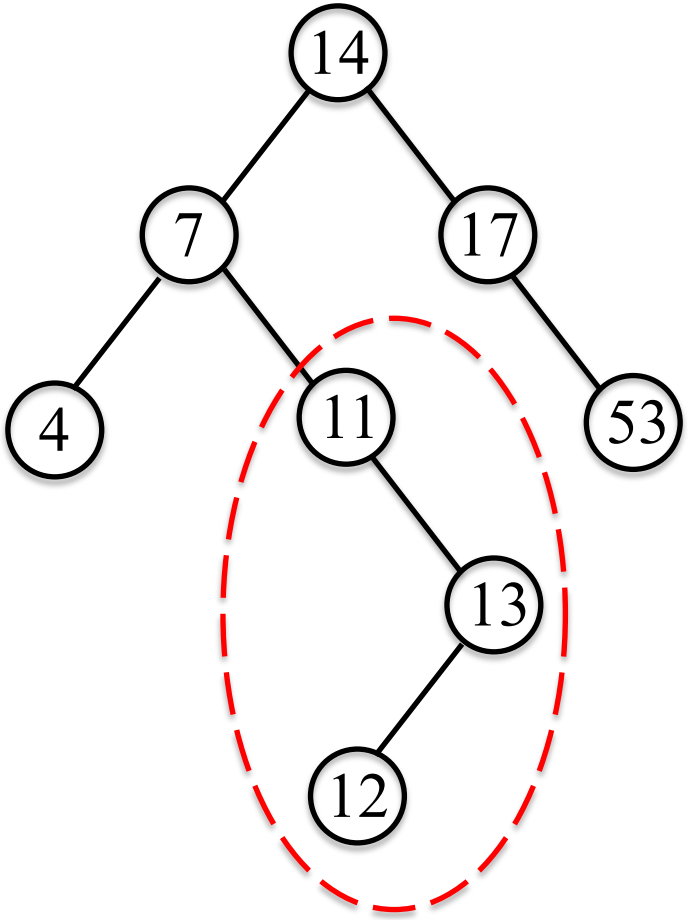
AVL Trees: Example

- Insert each of these number into the AVL tree in order: 14, 17, 11, 7, 53, 4, 13, 12



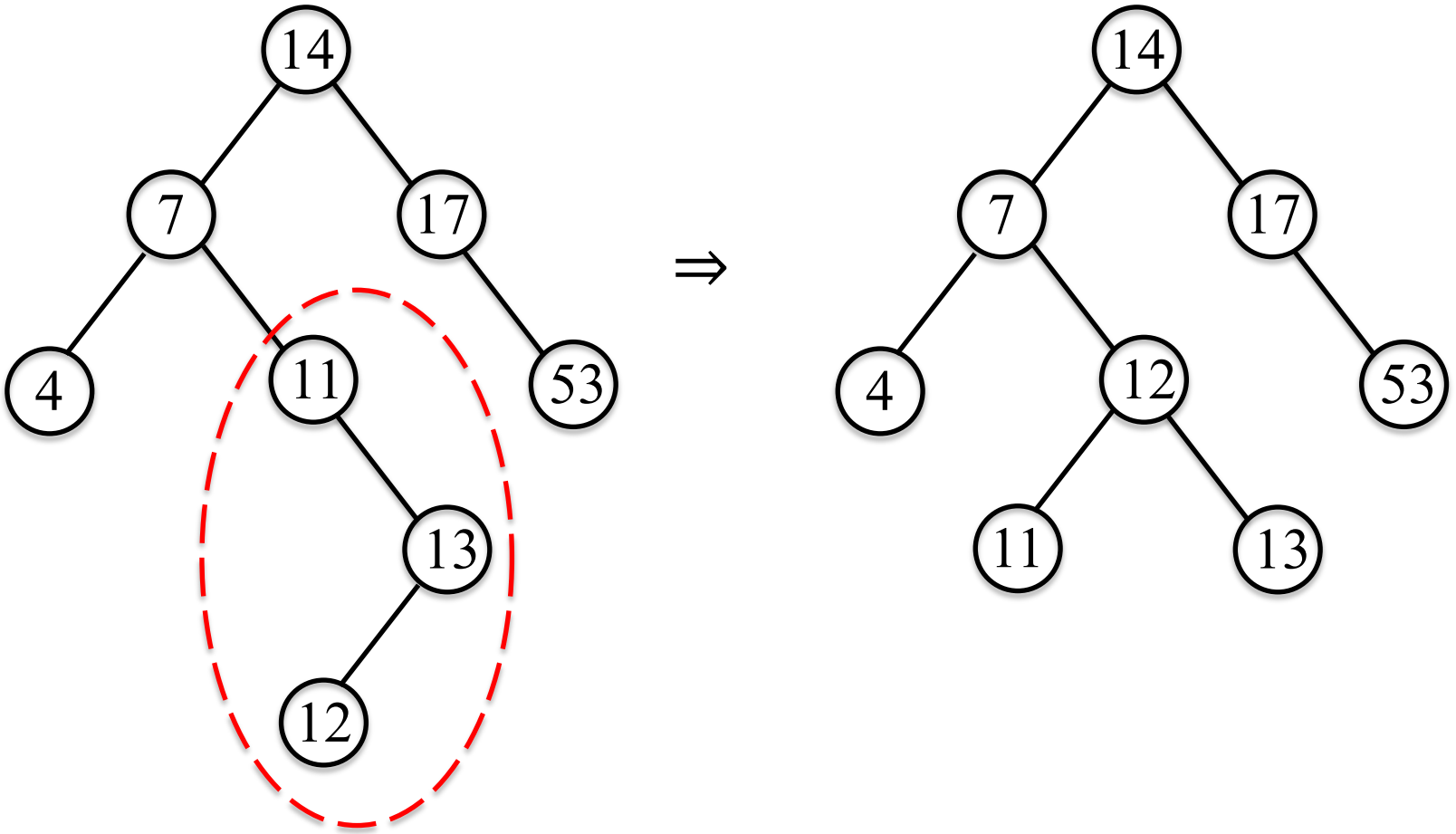
AVL Trees: Example

- Insert each of these number into the AVL tree in order: 14, 17, 11, 7, 53, 4, 13, 12

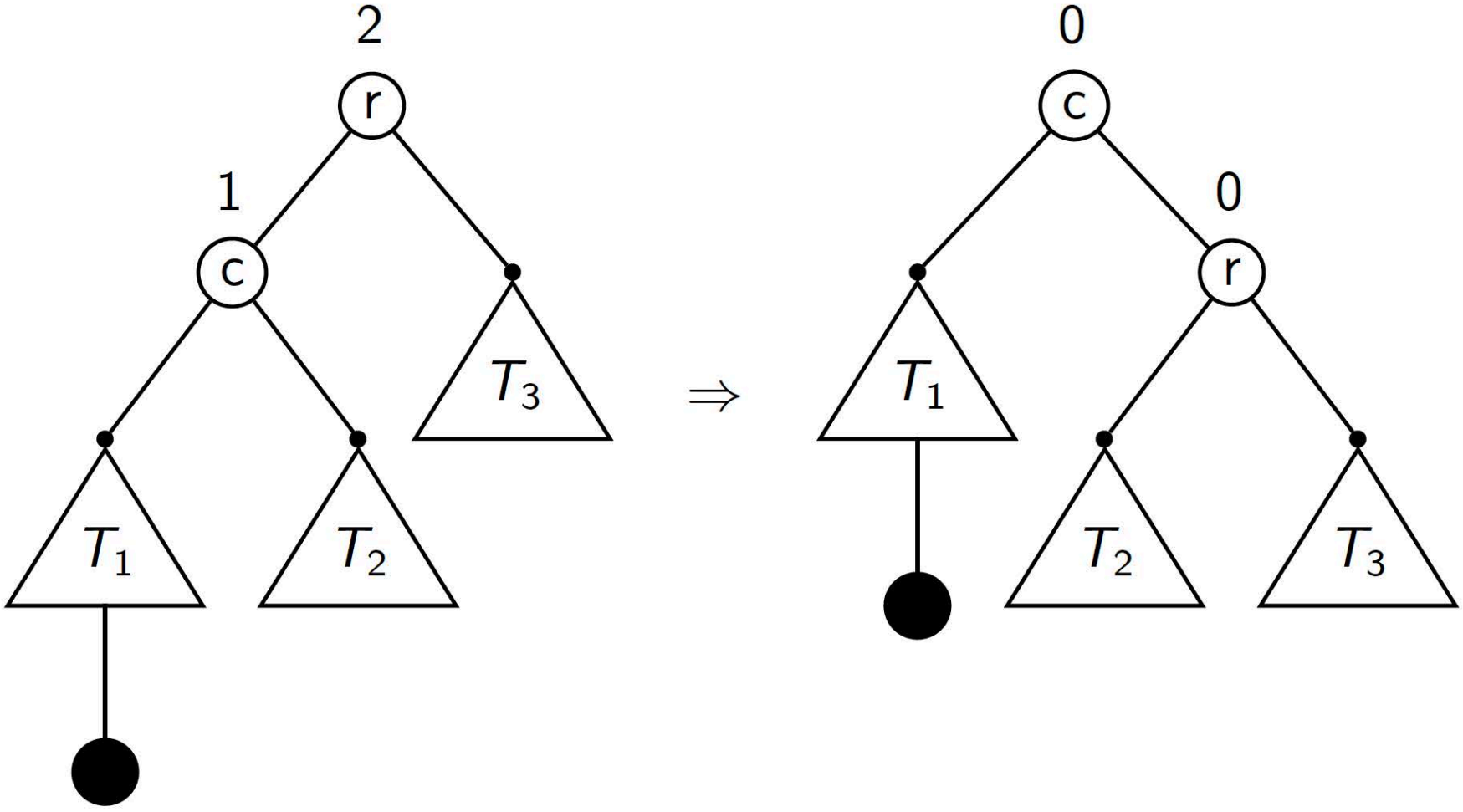


AVL Trees: Example

- Insert each of these number into the AVL tree in order: 14, 17, 11, 7, 53, 4, 13, 12

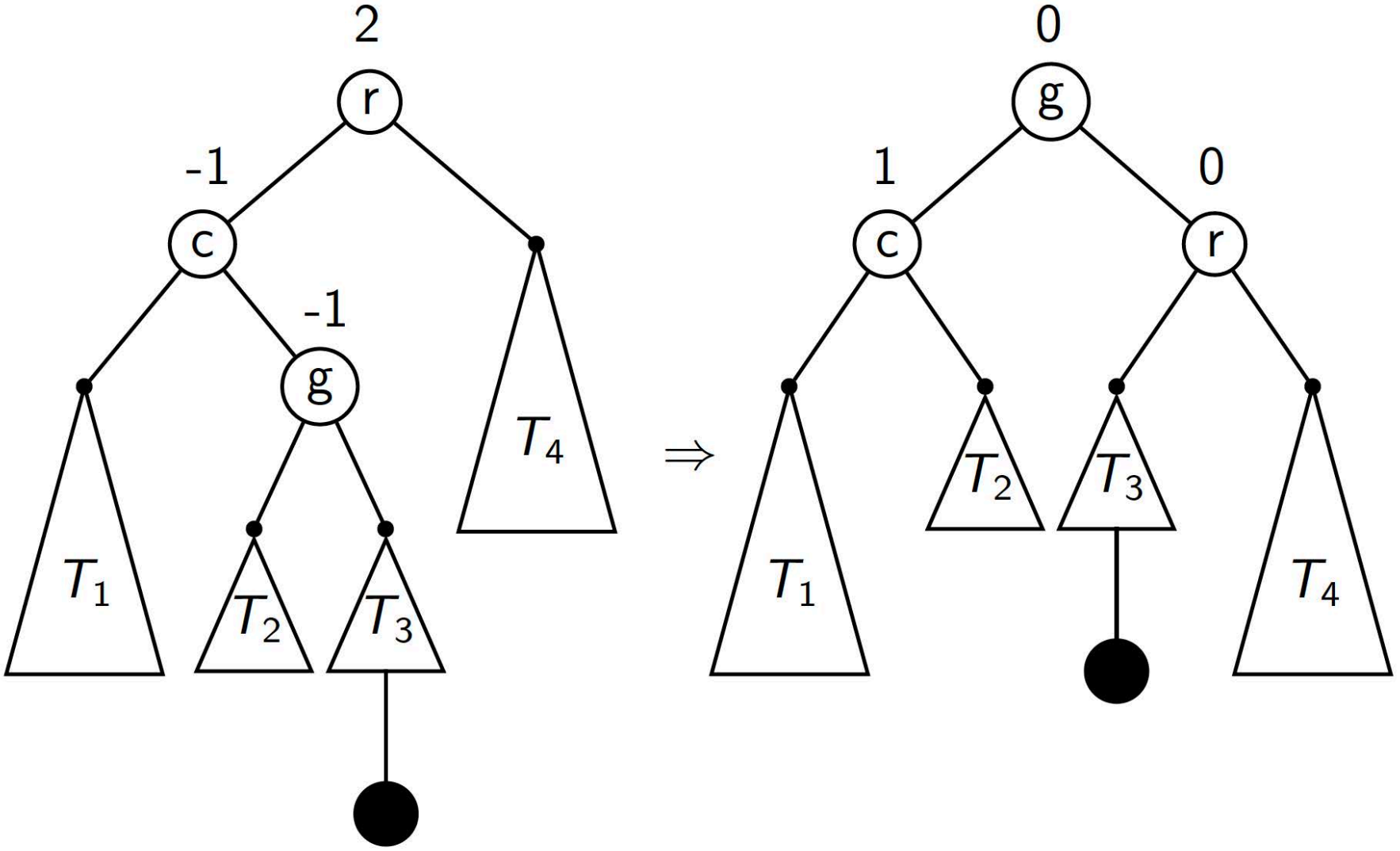


AVL Trees: The Single Rotation, Generally



- This shows an R-rotation; an L-rotation is similar.

AVL Trees: The Single Rotation, Generally



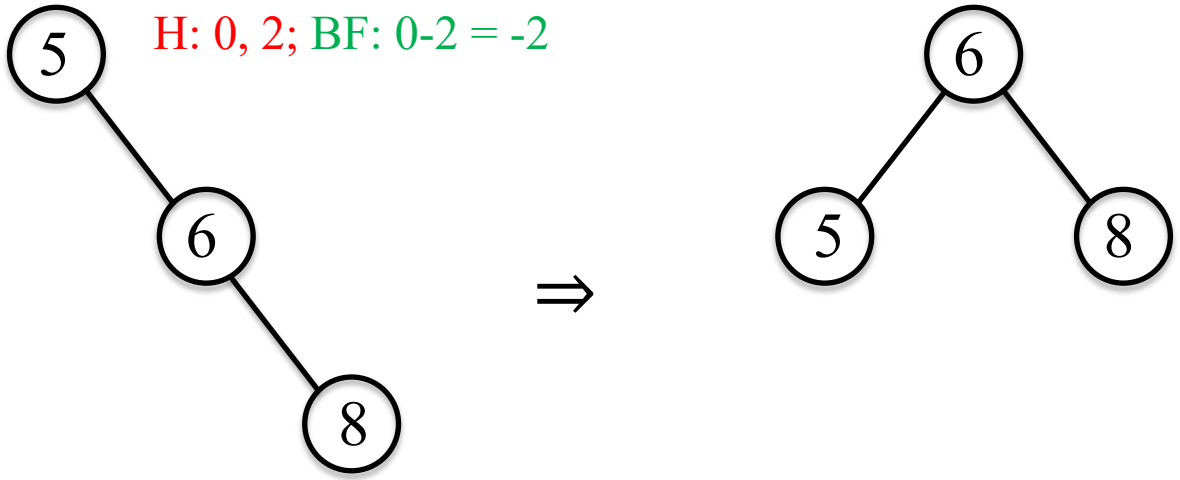
- This shows an LR-rotation; an RL-rotation is similar.

AVL Trees: Example

- Insert each of these number into the AVL tree in order: 5, 6, 8, 3, 2, 4, 7

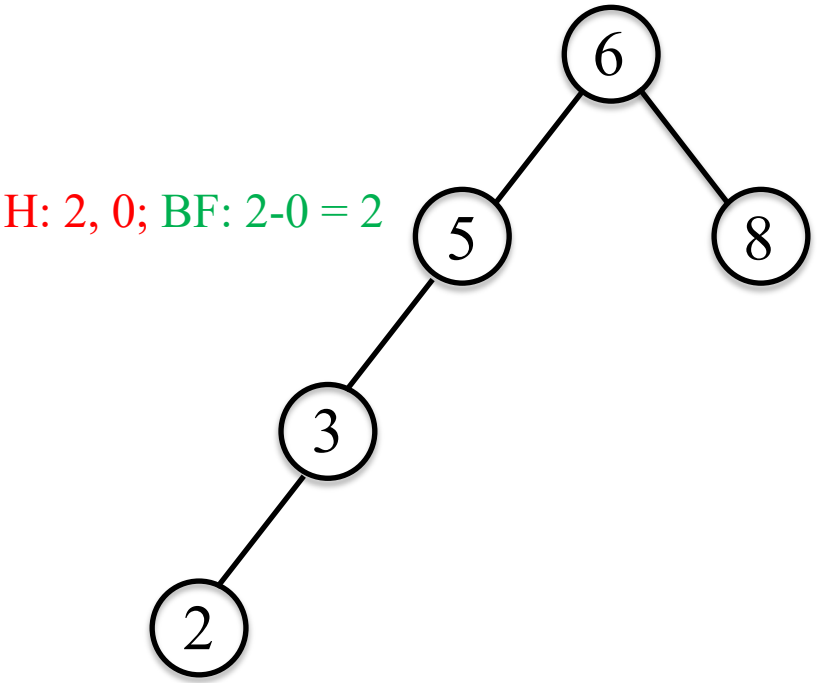
AVL Trees: Example

- Insert each of these number into the AVL tree in order: 5, 6, 8, 3, 2, 4, 7



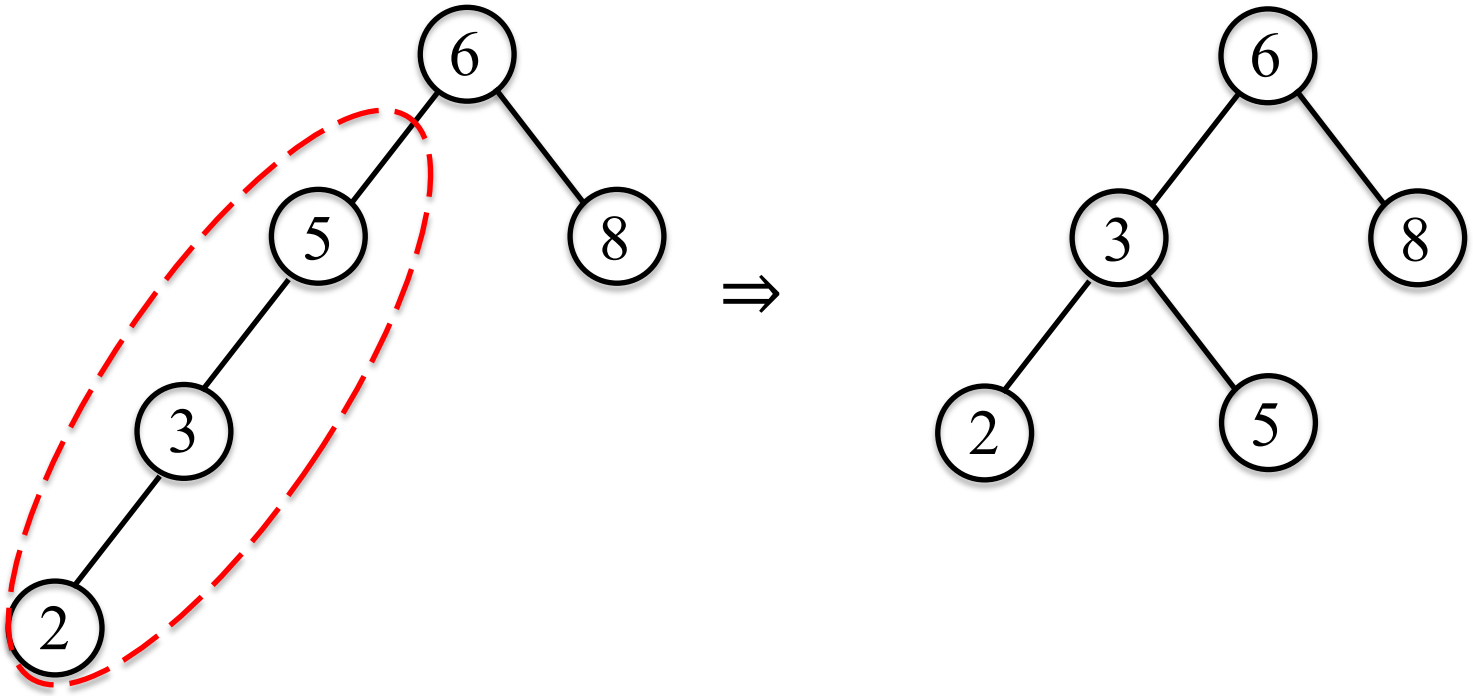
AVL Trees: Example

- Insert each of these number into the AVL tree in order: 5, 6, 8, 3, 2, 4, 7



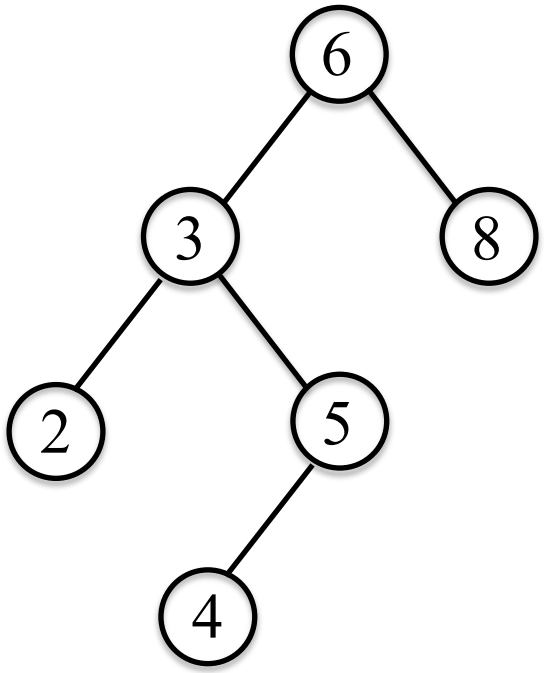
AVL Trees: Example

- Insert each of these number into the AVL tree in order: 5, 6, 8, 3, 2, 4, 7



AVL Trees: Example

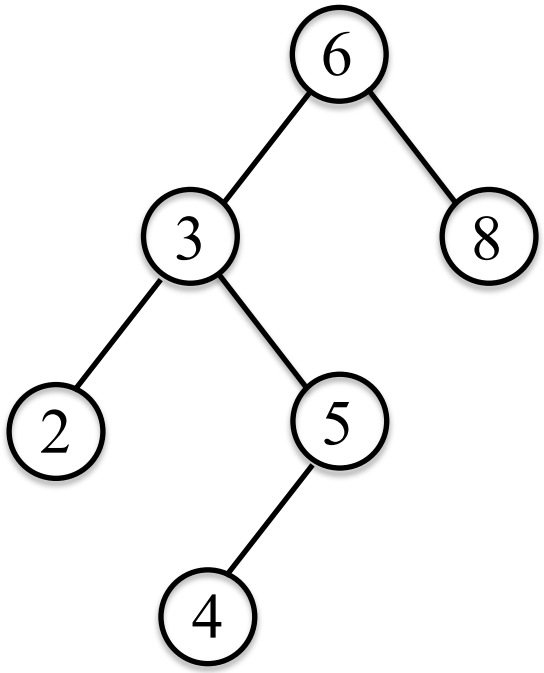
- Insert each of these number into the AVL tree in order: 5, 6, 8, 3, 2, 4, 7



AVL Trees: Example

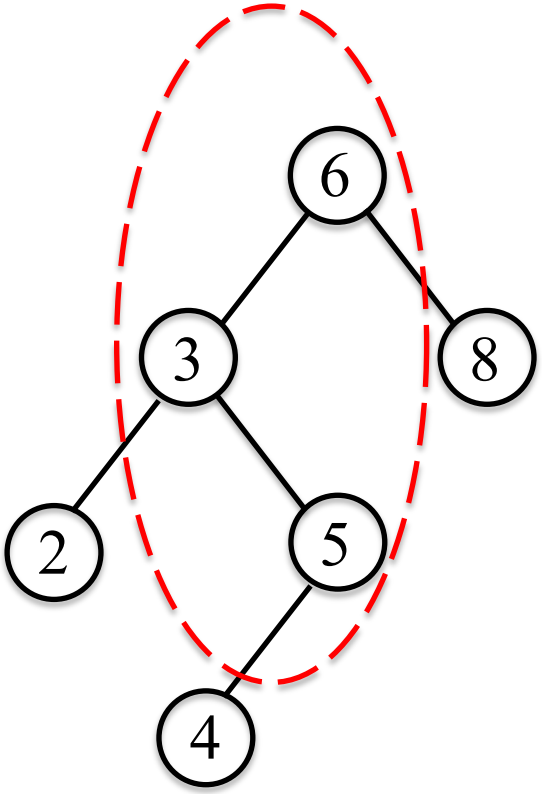
- Insert each of these number into the AVL tree in order: 5, 6, 8, 3, 2, 4, 7

H: 3, 1; BF: 3-1 = 2



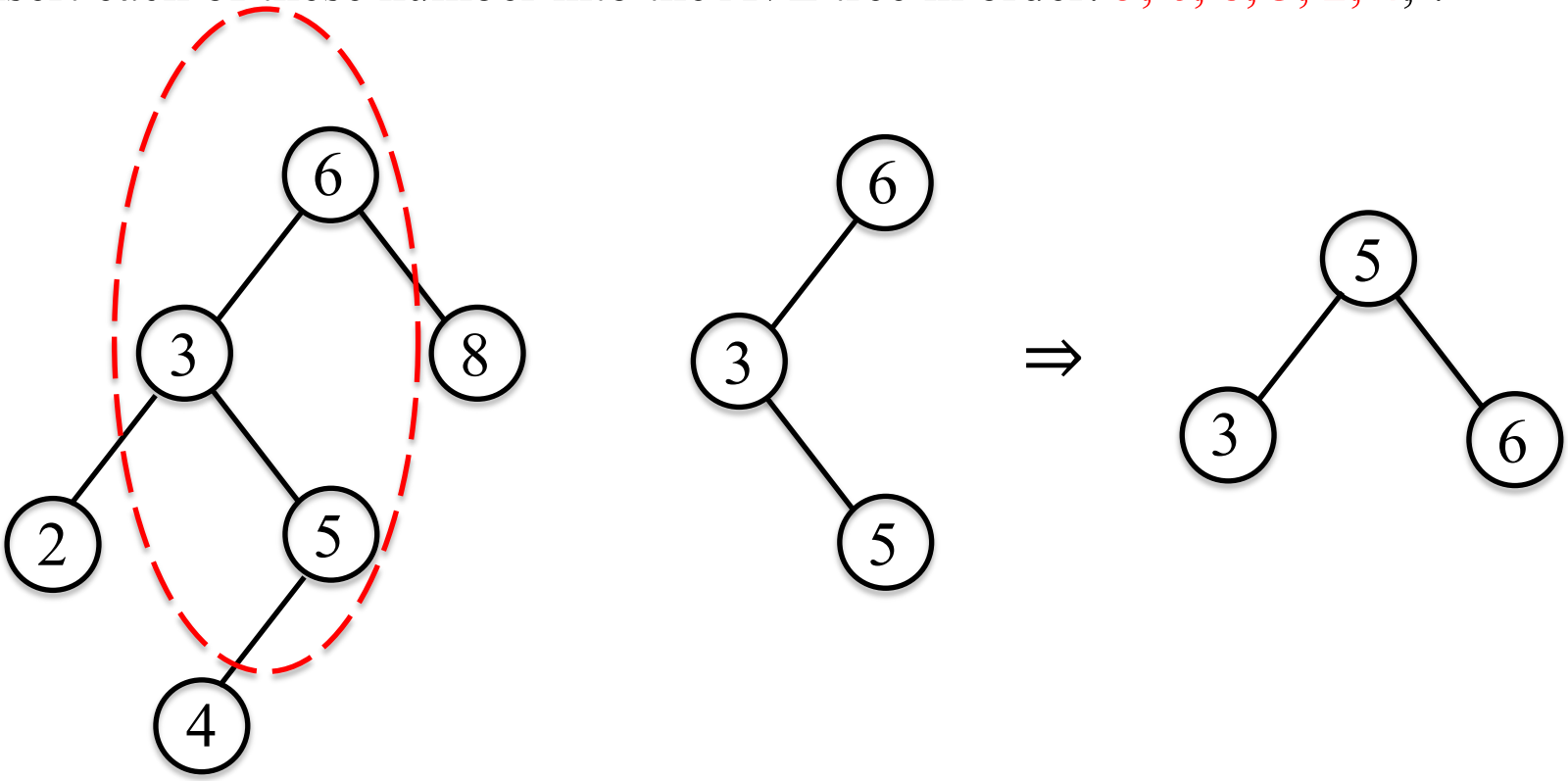
AVL Trees: Example

- Insert each of these number into the AVL tree in order: 5, 6, 8, 3, 2, 4, 7



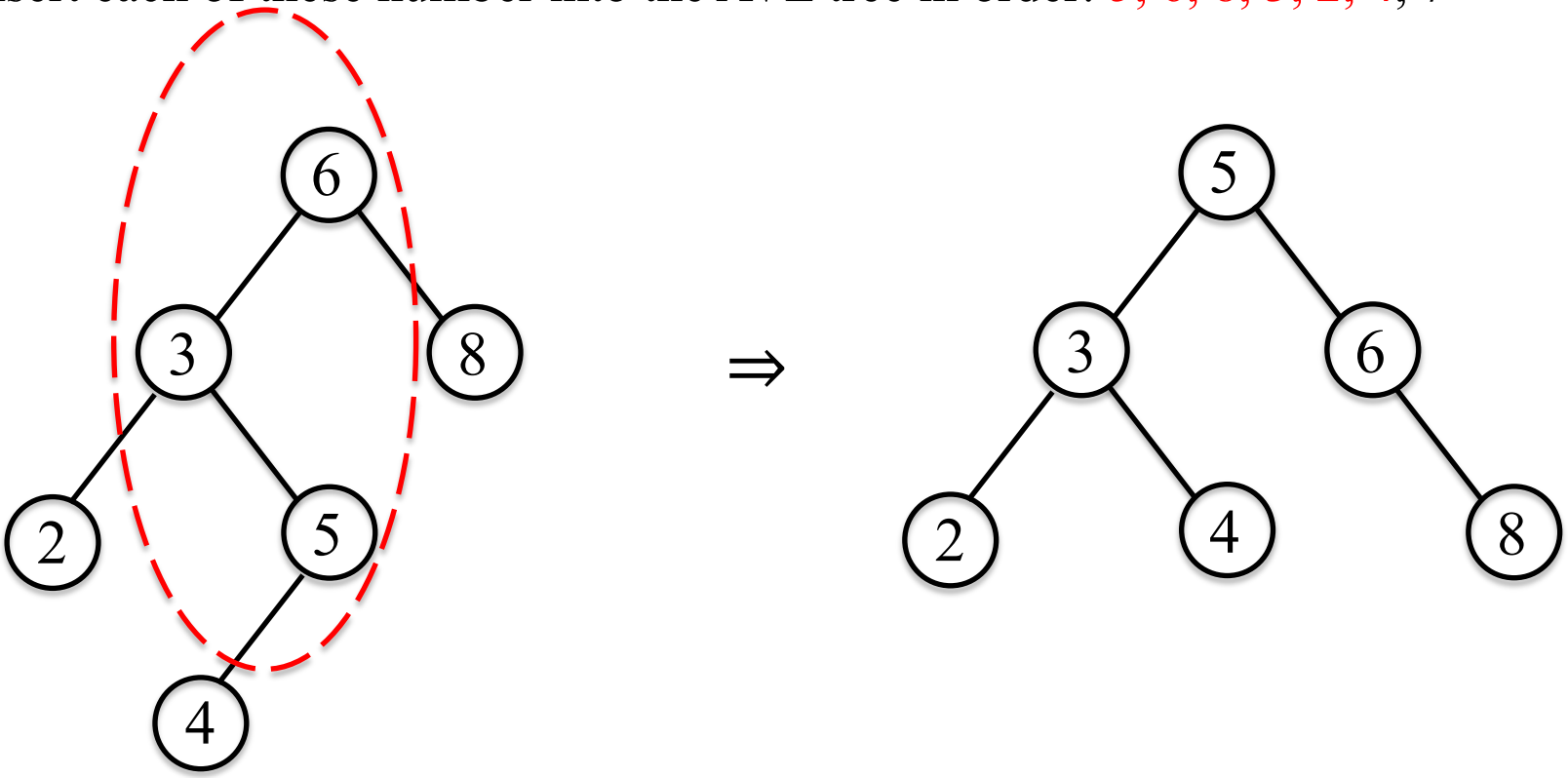
AVL Trees: Example

- Insert each of these number into the AVL tree in order: 5, 6, 8, 3, 2, 4, 7



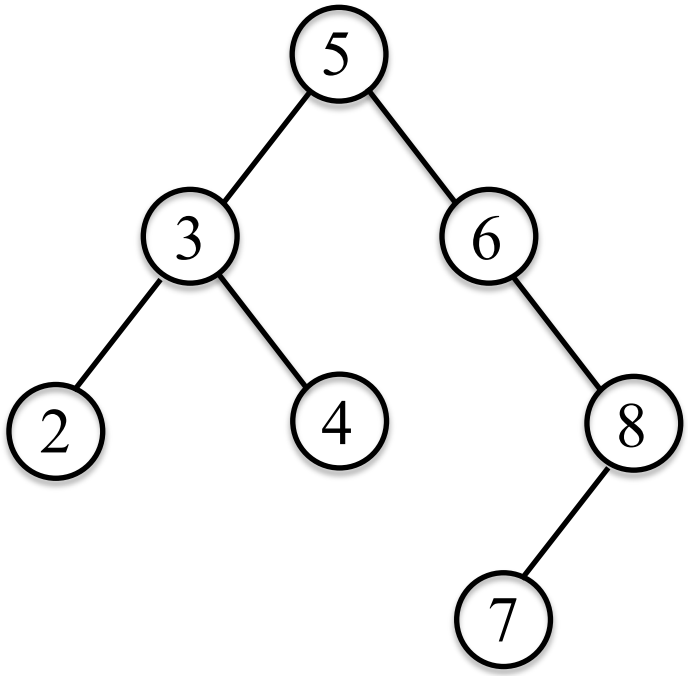
AVL Trees: Example

- Insert each of these number into the AVL tree in order: 5, 6, 8, 3, 2, 4, 7



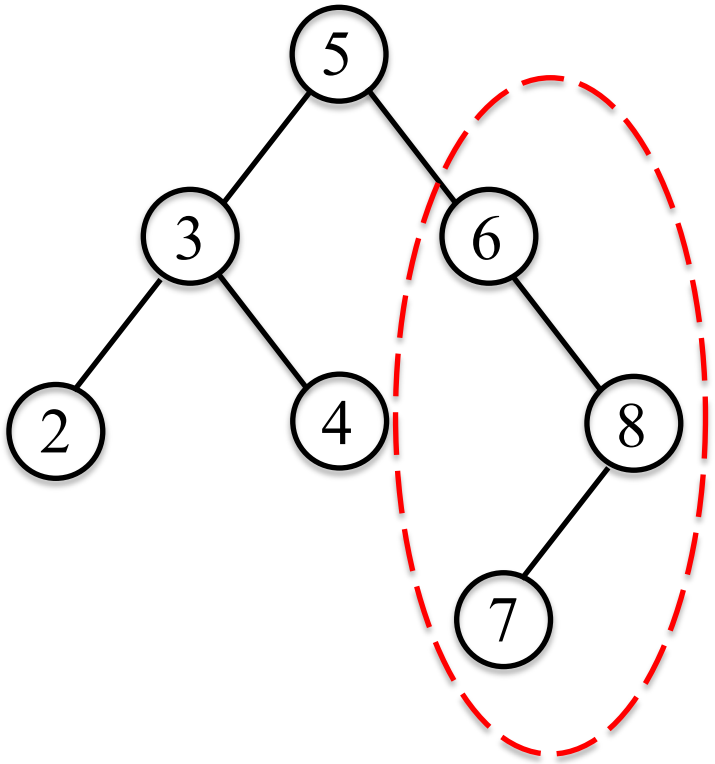
AVL Trees: Example

- Insert each of these number into the AVL tree in order: 5, 6, 8, 3, 2, 4, 7



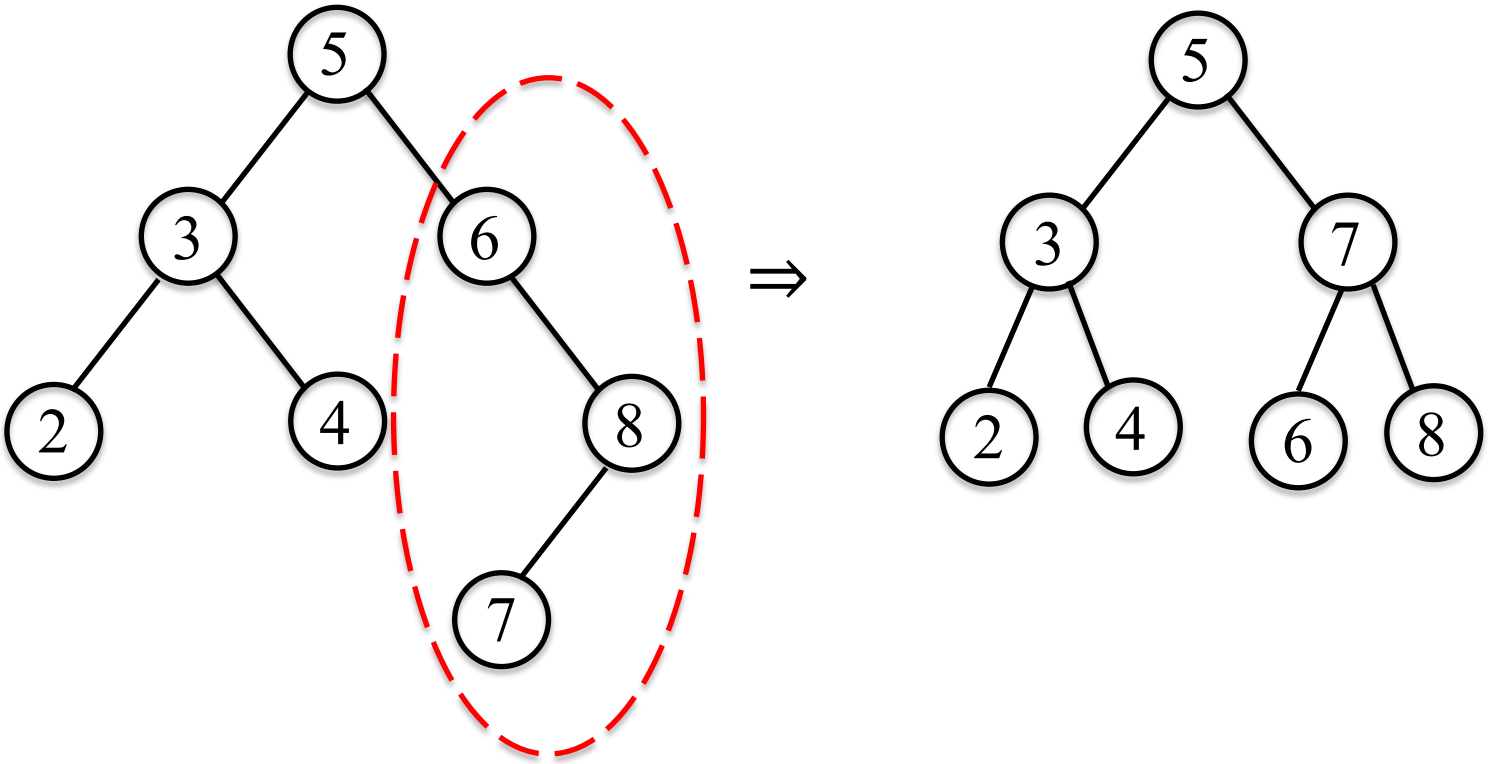
AVL Trees: Example

- Insert each of these number into the AVL tree in order: 5, 6, 8, 3, 2, 4, 7



AVL Trees: Example

- Insert each of these number into the AVL tree in order: 5, 6, 8, 3, 2, 4, 7

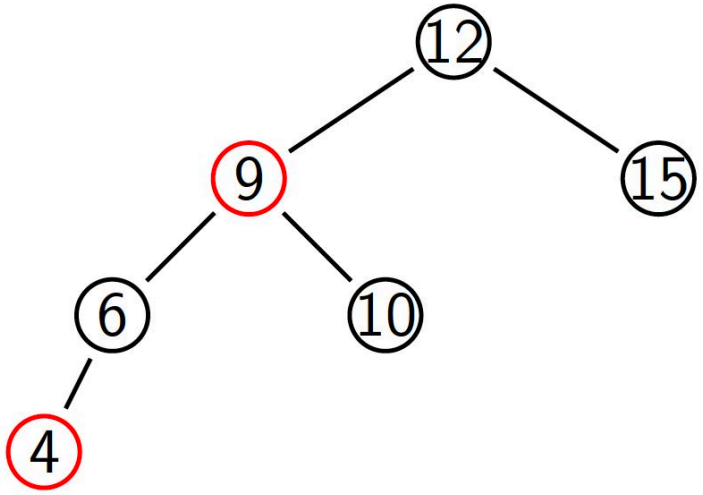


Properties of AVL Trees

- The notion of "balance" that is implied by the AVL condition is sufficient to guarantee that the depth of an AVL tree with n nodes is $\Theta(\log n)$.
- For random data, the depth is very close to $log_2 n$, the optimum.
- In the worst case, search will need at most 45% more comparisons than with a perfectly balanced BST.
- Deletion is harder to implement that insertion but also $\Theta(\log n)$.

Other Kinds of Balanced Trees

- A red-black tree is a BST with a slightly different concept of "balanced". Its nodes are coloured red of black, so that:
  - No red node has a red child.
  - Every path from the root to the fringe has the same number of black nodes.



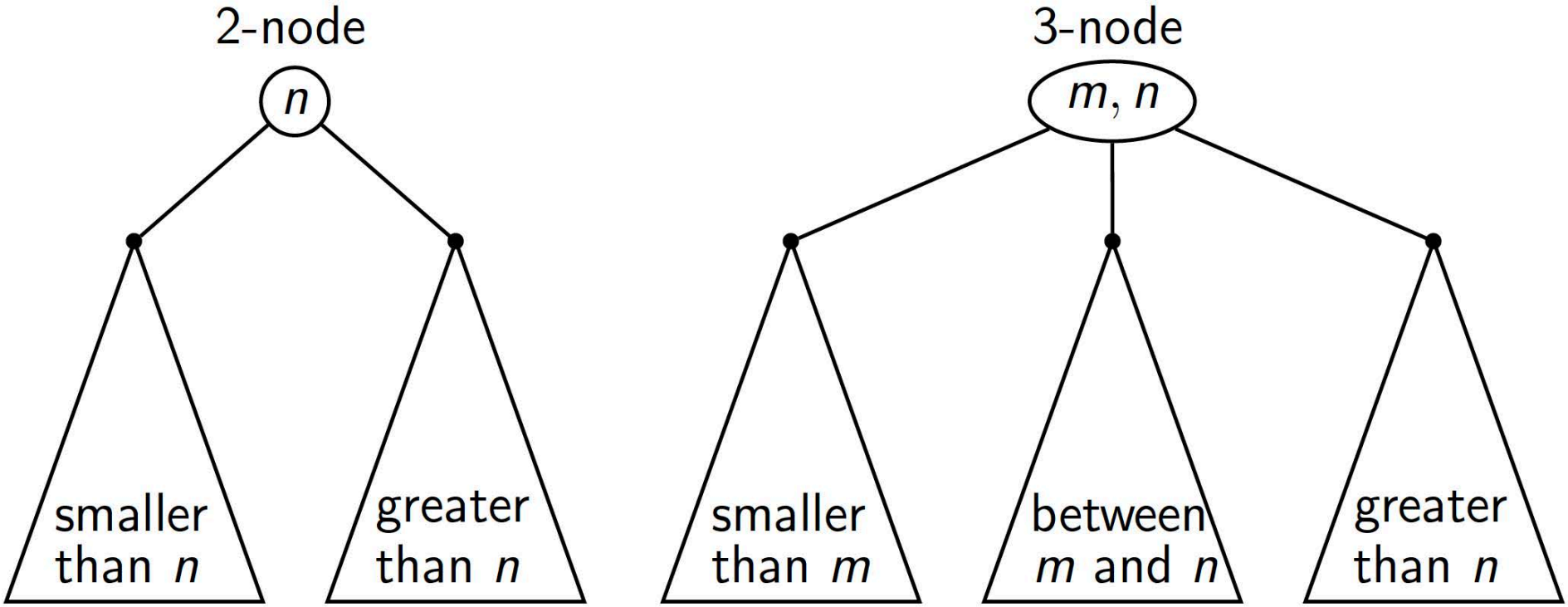
A worst-case red-black tree (the longest path is twice as long as the shortest path).

A splay tree is a BST which is not only self-adjusting, but also adaptive. Frequently accessed items are brought closer to the root, so their access becomes cheaper.

2-3 Trees

- 2-3 trees and 2-3-4 trees are search trees that allow more than one item to be stored in a tree node.
- As with BSTs, a node that holds a single item has (at most) two children.
- A node that holds two items (a so-called 3-node) has (at most) three children.
- And for 2-3-4 trees, a node that holds three items (a 4-node) has (at most) four children.
- This allows for a simple way of keeping search trees balanced.

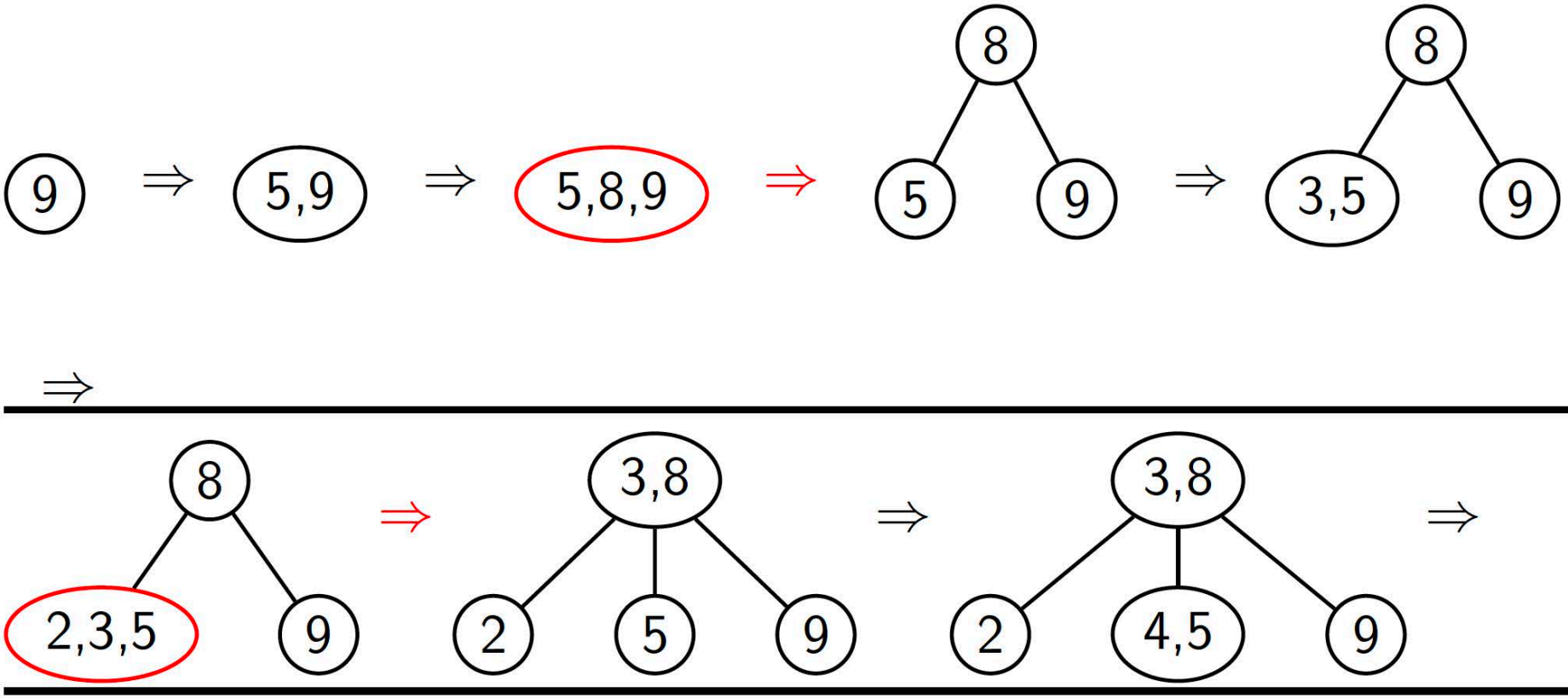
2-Nodes and 3-Nodes



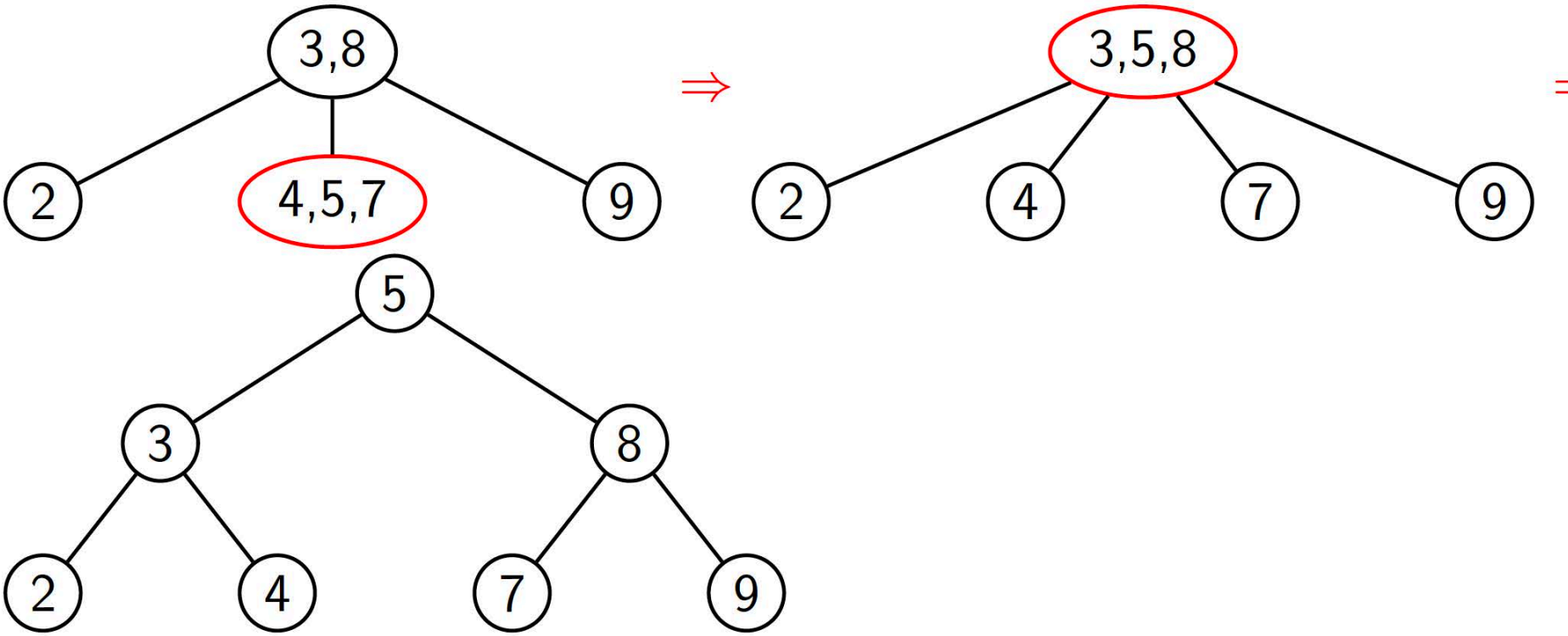
Insertion in a 2-3 Tree

- To insert a key k, pretend that we are searching for k.
- This will take us to a leaf node in the tree, where k should now be inserted; if the node we find there is a 2-node, k can be inserted without further ado.
- Otherwise we had a 3-node, and the two inhabitants, together with k, momentarily form a node with three elements; in sorted order, call them k_1, k_2 and k_3.
- We now split the node, so that k_1 and k_3 form their own individual 2-nodes. The middle key, k_2 is promoted to the parent node.
- The promotion may cause the parent node to overflow, in which case it gets split the same way. The only time the tree's height changes is when the root overflows.

Example: Build a 2-3 Tree from 9, 5, 8, 3, 2, 4, 7



Example: Build a 2-3 Tree from 9, 5, 8, 3, 2, 4, 7

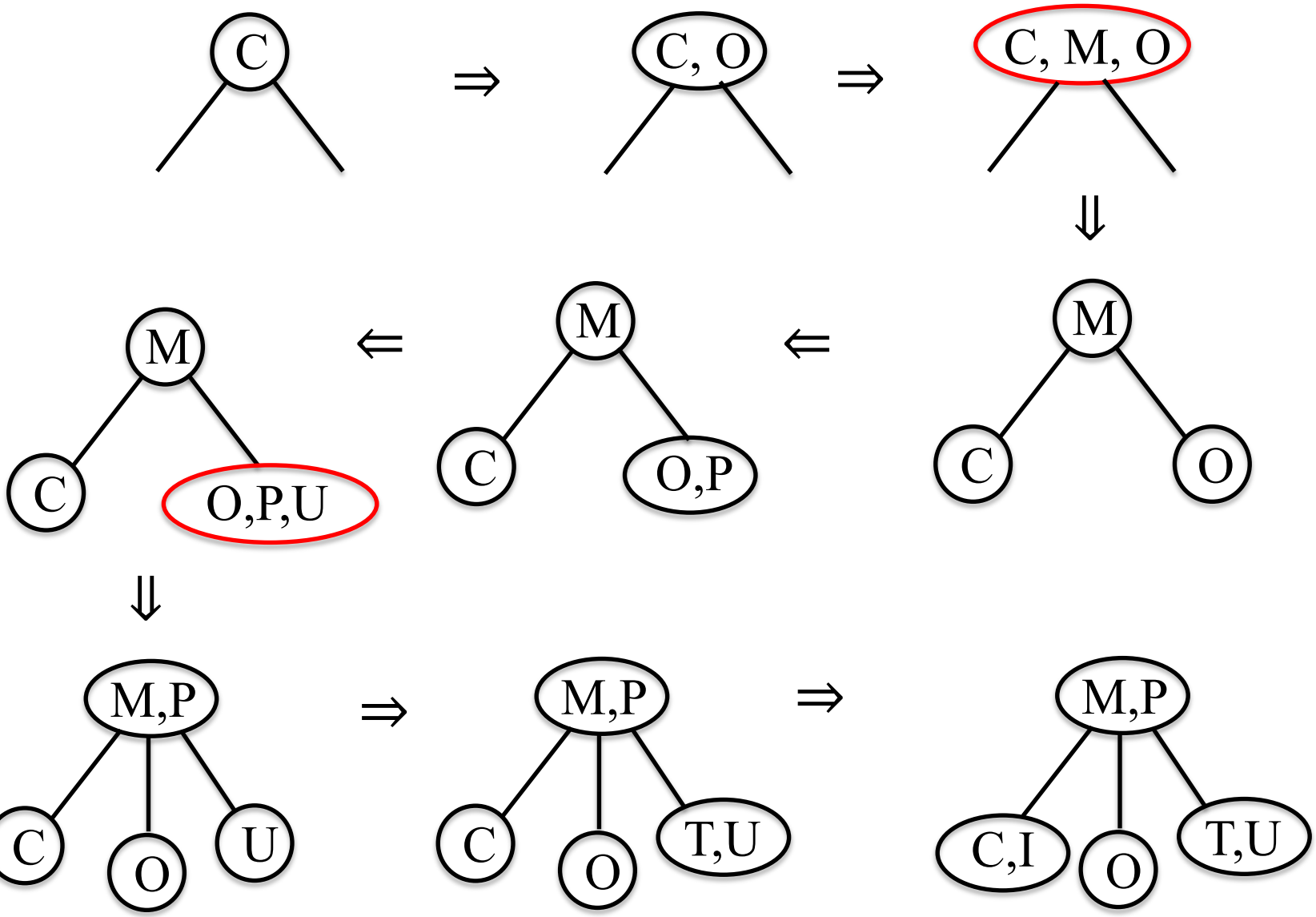


Exercise: 2-3 Tree Construction: C, O, M, P, U, T, I, N, G

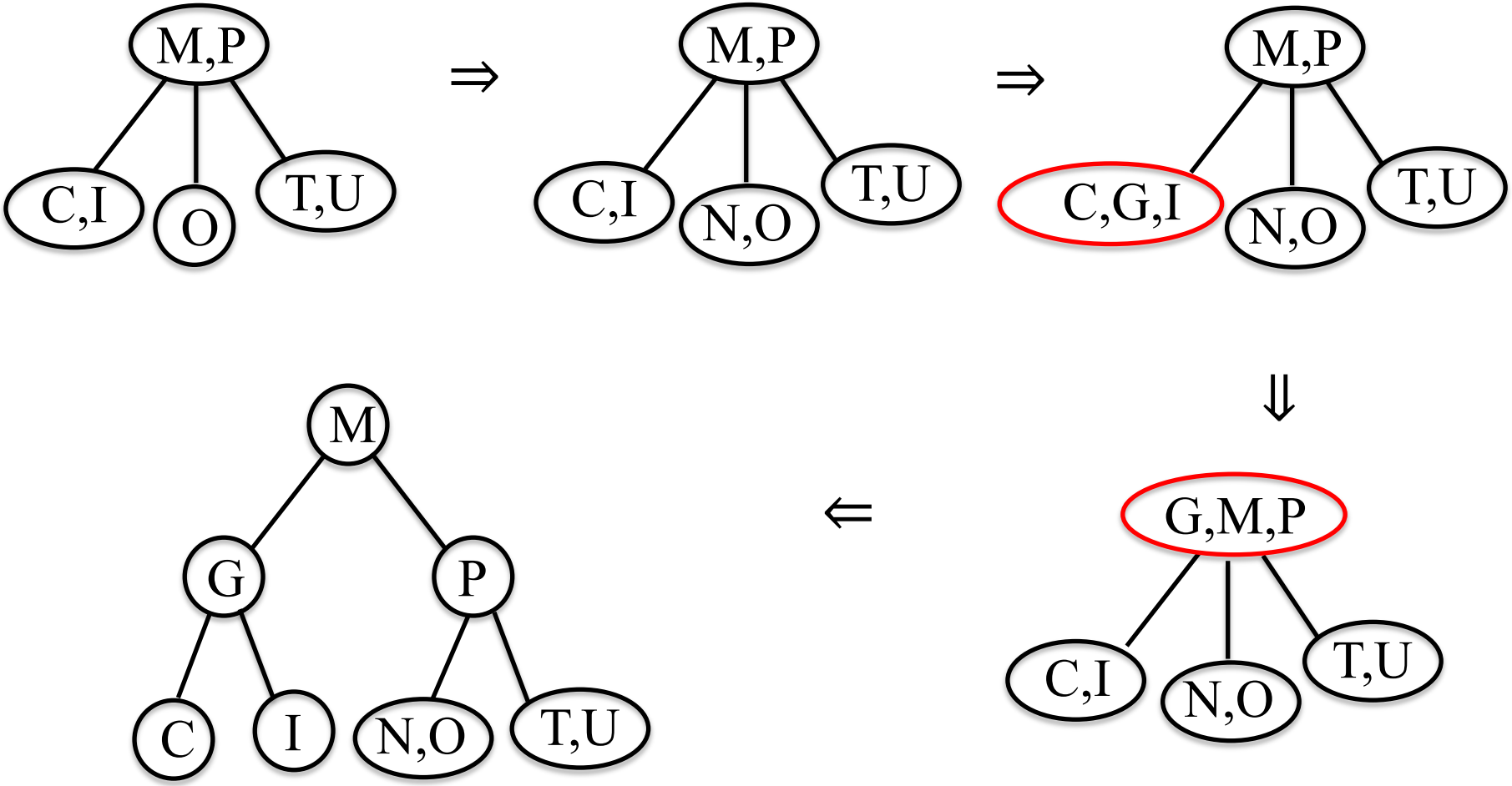
- Build the 2-3 tree that results from inserting these keys, in the given order, into an initially empty tree:

C, O, M, P, U, T, I, N, G

Exercise: 2-3 Tree Construction: C, O, M, P, U, T, I, N, G



Exercise: 2-3 Tree Construction: C, O, M, P, U, T, I, N, G



2-3 Tree Analysis

- Worst case search time results when all nodes are 2-nodes.
- The relation between the number n of nodes and the height h is:

$$n = 1 + 2 + 4 + \cdots + 2^h = 2^{h+1} - 1$$

- That is, $log_2(n + 1) = h + 1$.
- In the best case, all nodes are 3-nodes:

$$n = 2 + 2\times3 + 2\times3^2 + \cdots + 2\times3^h = 3^{h+1} - 1$$

- That is, $log_3(n + 1) = h + 1$.

- Hence we have $log_3(n + 1) - 1 \ \leq h \leq log_2(n + 1) - 1$.
- Useful formula: $\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1}$, for $a \neq 1$.

Coming Up Next

- How to buy time, by spending a bit more space (Levitin 7.1 & 7.2).