Introduction to AVL Trees

Unit 6

CC4 Data Structures and Algorithms

Christine T. Gonzales

Table of Contents

Introduction to AVL Trees

Review: Binary Search Trees | The problem with binary search trees | What is an AVL tree?

Review: Binary Search Trees

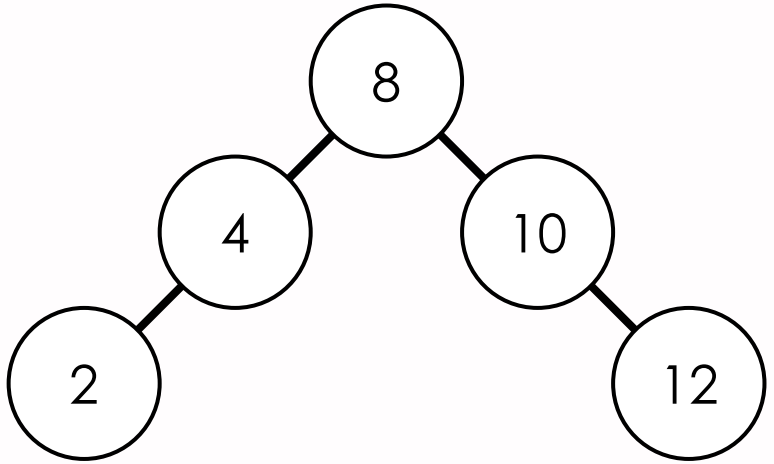
- Each parent node has a maximum of two child nodes
  - Left child: lesser value than the parent node
  - Right child: higher value than the parent node
- Each tree has a tree height
  - Number of levels from the root node
  - Shorter tree height is usually more efficient in searching

Review: Binary Search Trees

BST values: 8, 4, 2, 10, 12

Tree height: 2

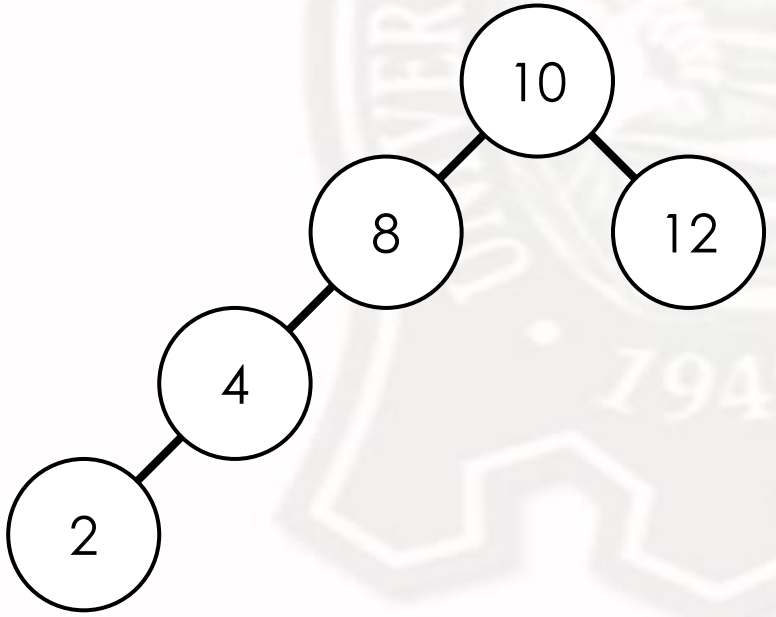
Type of tree: balanced



BST values: 10, 8, 4, 2, 12

Tree height: 3

Type of tree: unbalanced / skewed



The Problem with Binary Search Trees

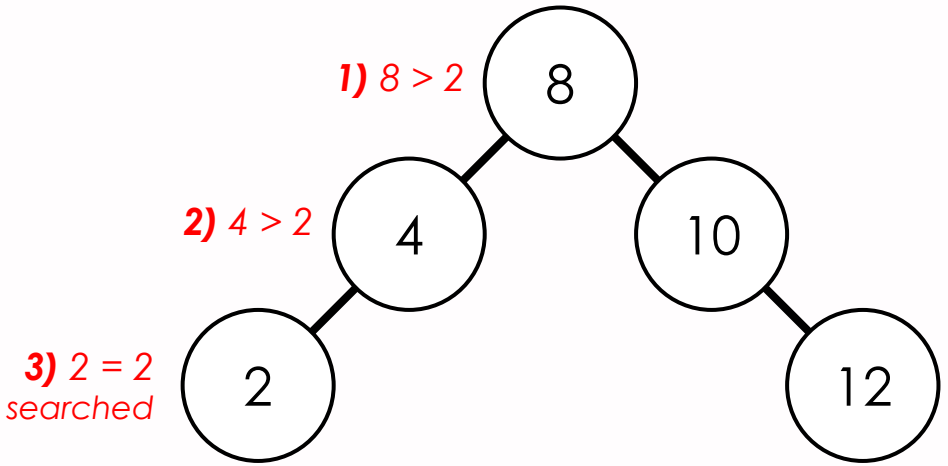
- Unbalanced / skewed BSTs can be less efficient:
  - Binary search trees are made to make searching more efficient
  - The higher the tree height, the more levels
- Balanced BSTs are usually more efficient:
  - Tends to have a lesser tree height

The Problem with Binary Search Trees

Example: Search '2' from the tree

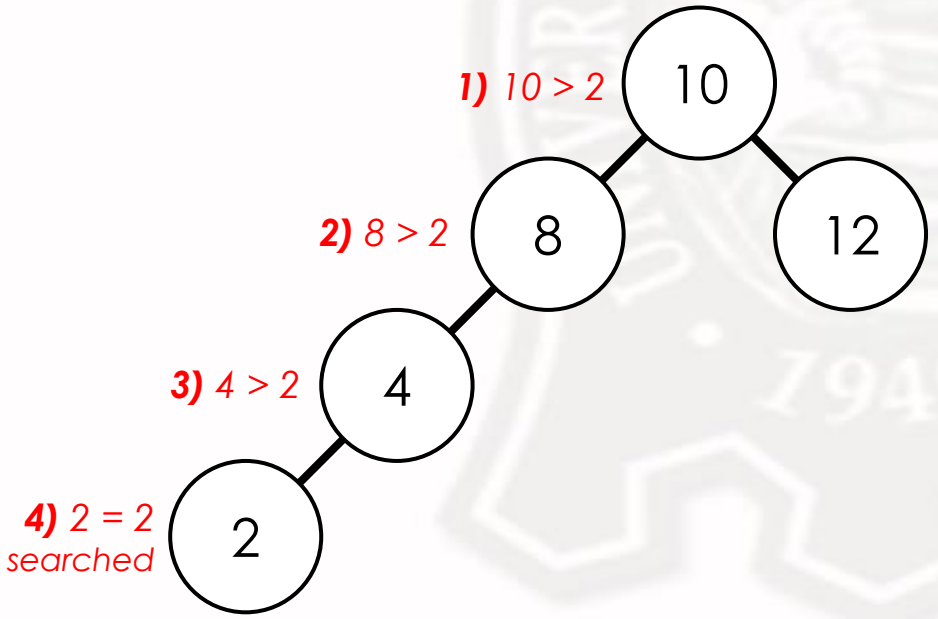
Tree height: 2

Number of comparisons: 3



Tree height: 3

Number of comparisons: 4



Introduction to AVL Trees

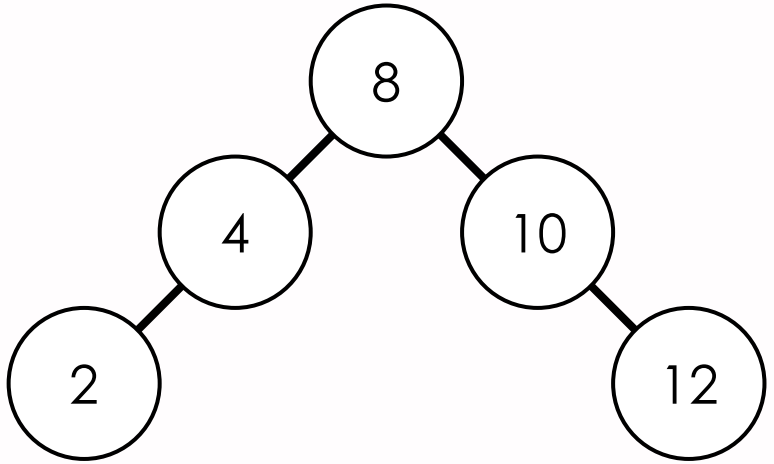
- Short for Adelson-Velsky and Landis tree
- Self-balancing binary search tree
- Determined by tree
  - Difference of tree height of the left subtree and the right subtree must be 0 or 1
  - When subtracting the heights, get the absolute value
  - If difference is higher than 1, then it is subject for rotation

Introduction to AVL Trees

Left subtree height: 2

Right subtree height: 2

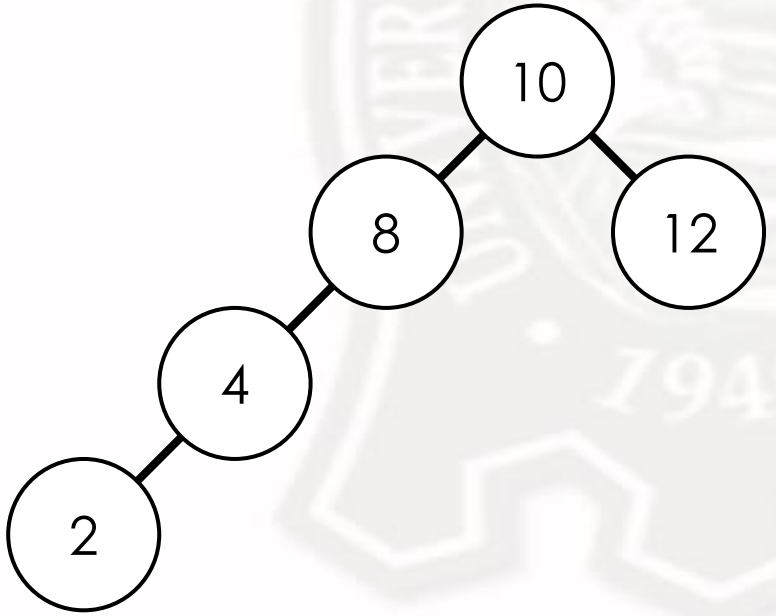
2 – 2 = /0/ = **0 ∴ an AVL tree**



Left subtree height: 3

Right subtree height: 1

3 – 1 = /2/ = **2 ∴ not an AVL tree**



Actions in AVL Trees

- Searching, Insertion, and Deletion
  - Steps are the same as a binary search tree
  - Rotation may be done after insertion or deletion
- Rotation
  - Done if the tree is considered as unbalanced
  - Ensure that when rotating, values still follow the rules of BSTs

AVL Tree Rotations

Identifying problems and violations | Left rotation | Right rotation | Left-right rotation | Right-left rotation

Identifying Problems and Violations

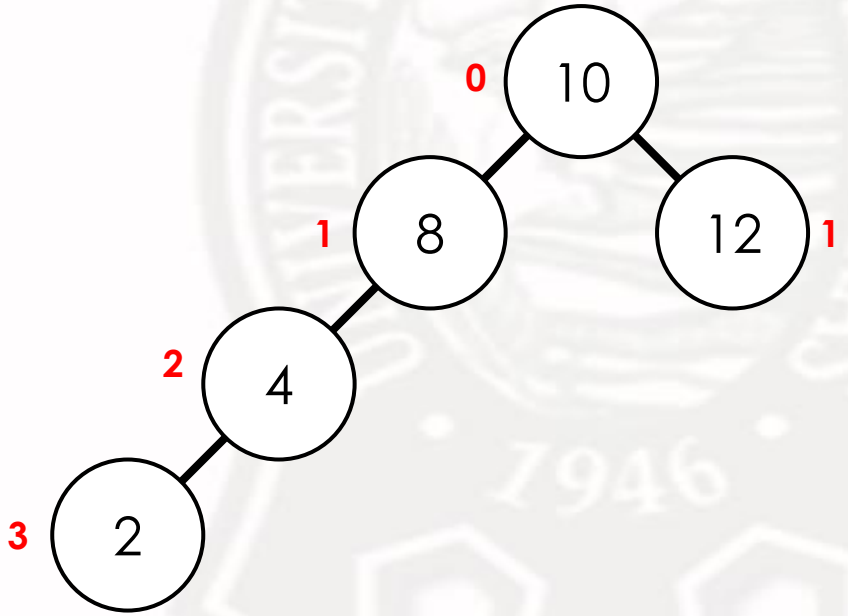
- When doing rotations, it is necessary to determine the items that are the cause of the problem and the steps necessary to fix it
- Problem
  - Node that causes the imbalance of the AVL tree
  - Usually a leaf node
- Violation
  - Node that is subjected to rotation
  - Usually the grandfather of the problem node

Identifying Problems and Violations

- Identify left and right subtree height
- Compute for the difference in subtree heights
- Determine the subtree with the longer tree height
- Identify the problem
- Identify the violation
- Determine the type of rotation necessary

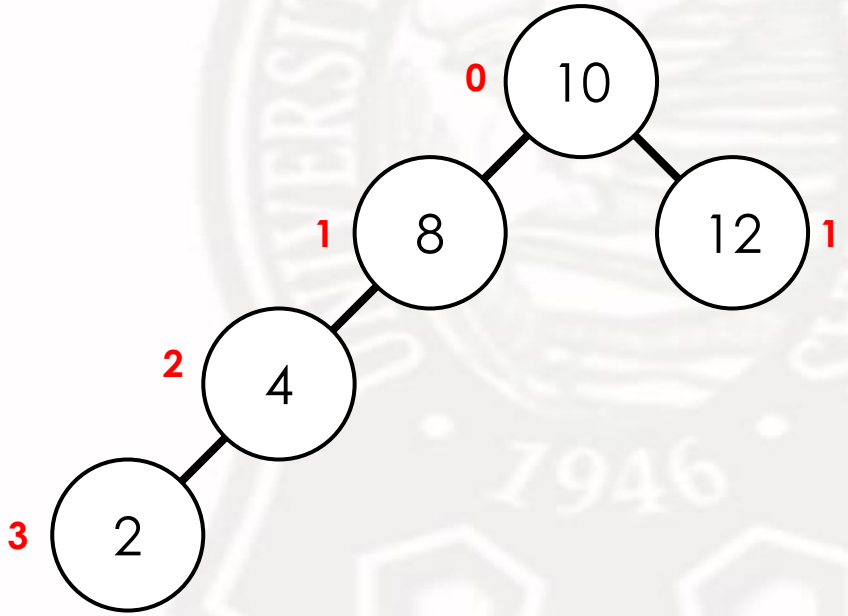
Identifying Problems and Violations

- ***Identify left and right subtree height***
  - Root node is counted as 0
  - Left subtree height (LSH) = 3
  - Right subtree height (RSH) = 1



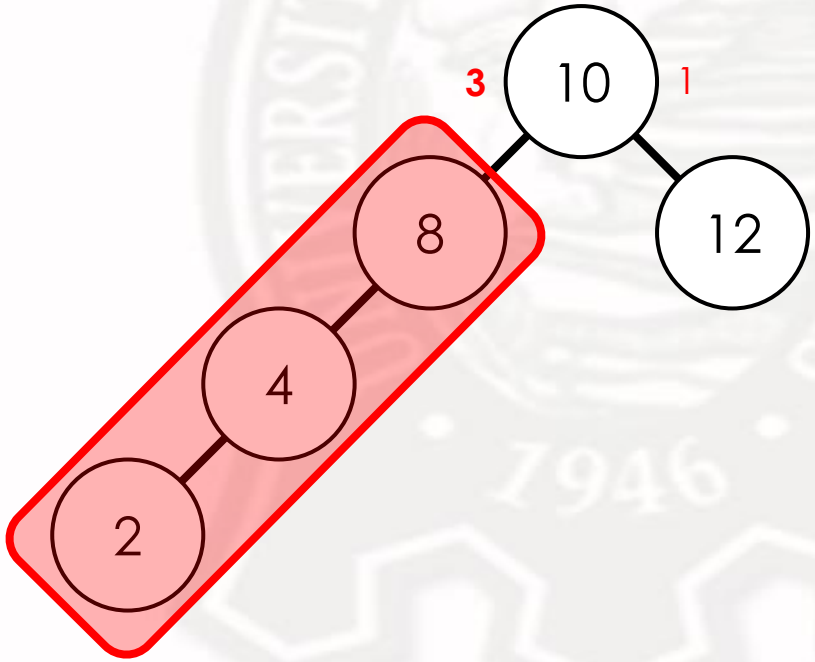
Identifying Problems and Violations

- ***Compute for the difference in subtree heights***
  - Difference = /LSH – RSH/
  - 3 – 1 = /2/ = 2
  - 2 is greater than 1 or 0; therefore it is unbalanced
  - Unbalanced BSTs are subject to rotation



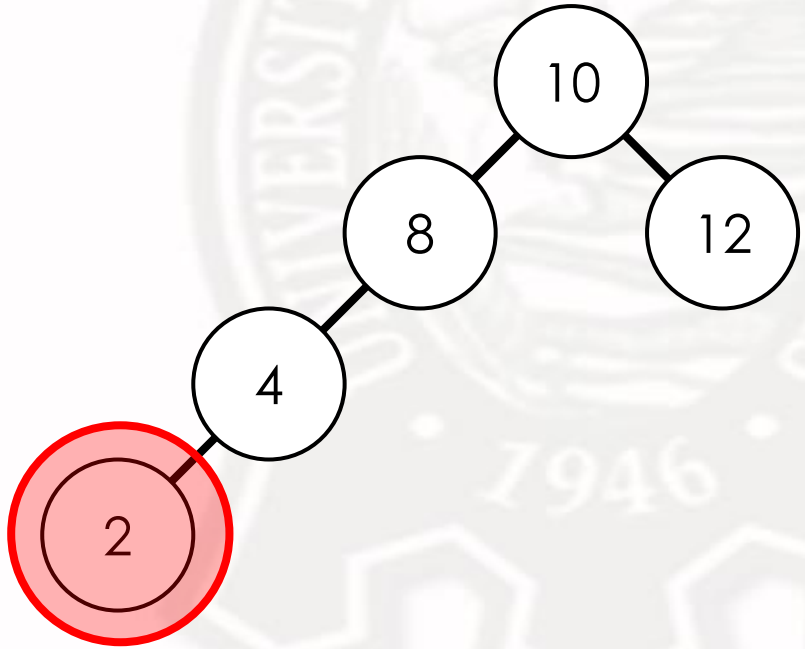
Identifying Problems and Violations

- Determine the subtree with the longer tree height
  - **Left subtree height = 3**
  - Right subtree height = 1



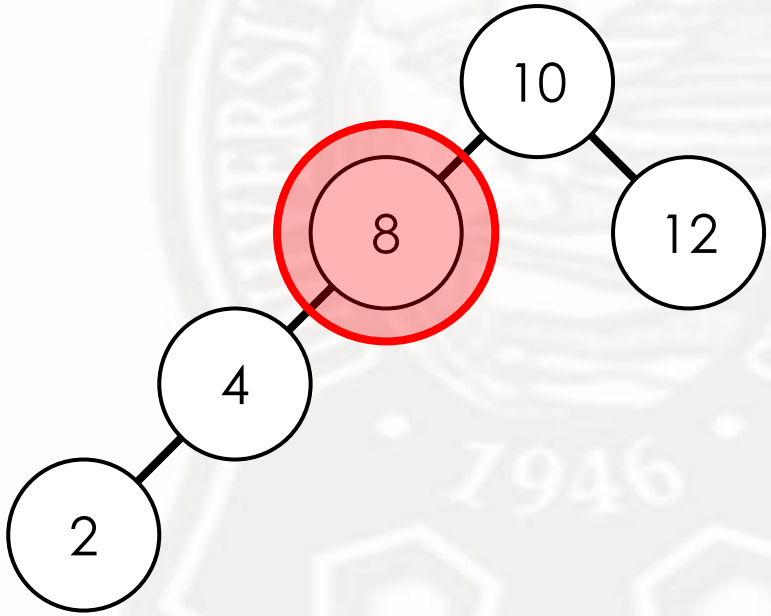
Identifying Problems and Violations

- ***Identify the problem***
  - Problem should be on the subtree with longer tree height (left subtree)
  - Problem is usually the leaf node
  - Problem node = 2



Identifying Problems and Violations

- Identify the violation
  - Violation is usually 2 nodes away from the leaf node
  - AKA “grandparent node”
  - Violation node = 8
- Determine the type of rotation necessary

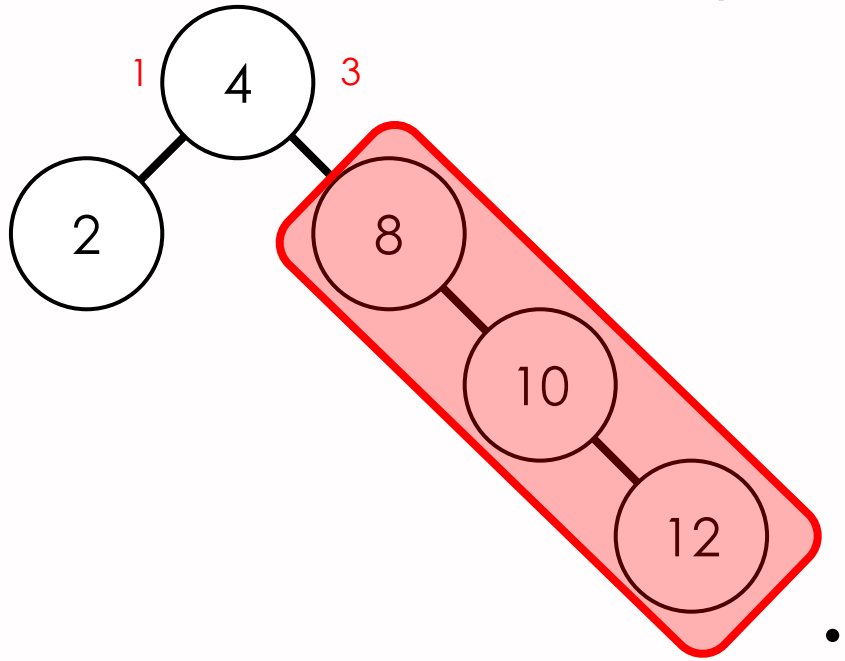


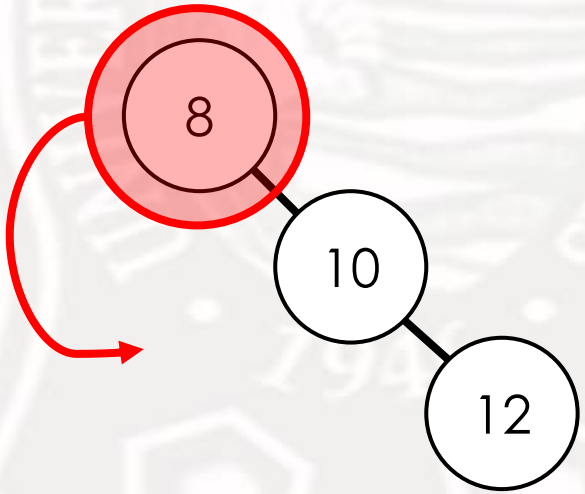
Common AVL Tree Rotations

- Left rotation (LL)
- Right rotation (RR)
- Left-right rotation (LR)
- Right-left rotation (RL)

Left Rotation

- Rotate the grandparent node to the left, becoming the left child node of the parent node

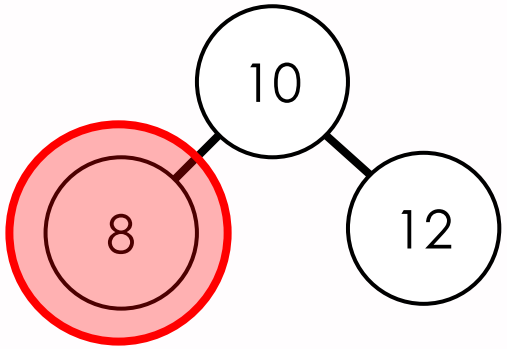


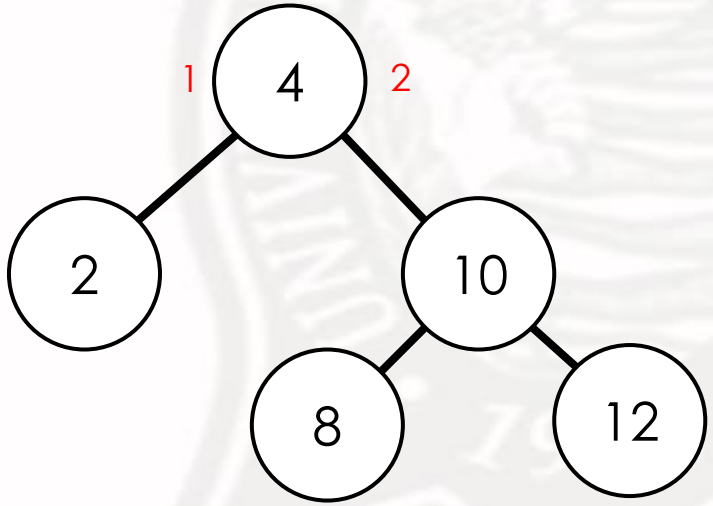


- Left subtree height: 1
- Right subtree height: 3
- 1 – 3 = /-2/ = 2 ∴ NOT an AVL tree

Left Rotation

- Rotate the grandparent node to the left, becoming the left child node of the parent node

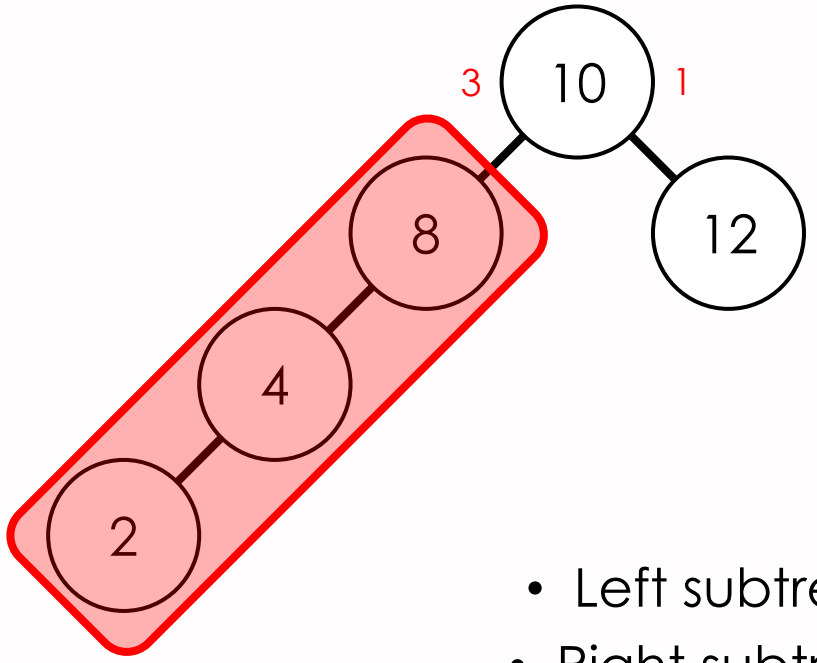


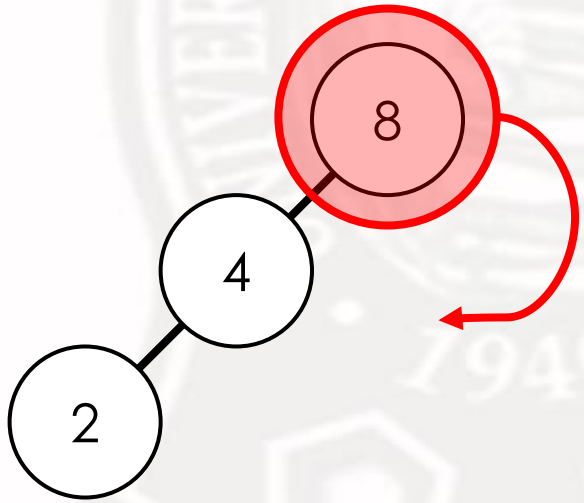


- Left subtree height: 1
- Right subtree height: 2
- 1 – 2 = /-1/ = 1 ∴ an AVL tree

Right Rotation

- Rotate the grandparent node to the right, becoming the right child node of the parent node

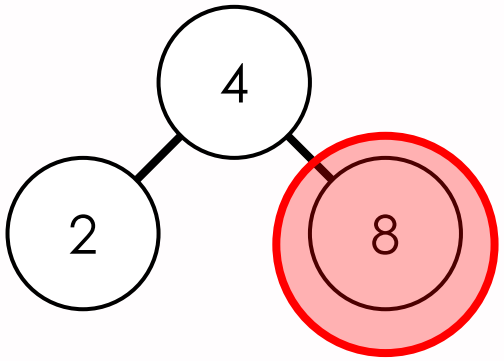


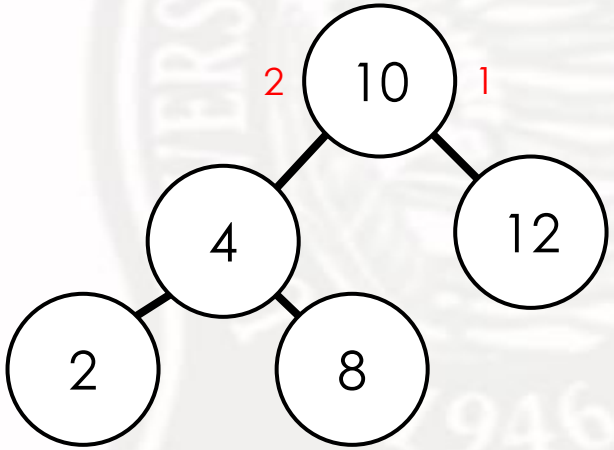


- Left subtree height: 3
- Right subtree height: 1
- 3 – 1 = /2/ = 2 ∴ NOT an AVL tree

Right Rotation

- Rotate the grandparent node to the right, becoming the right child node of the parent node

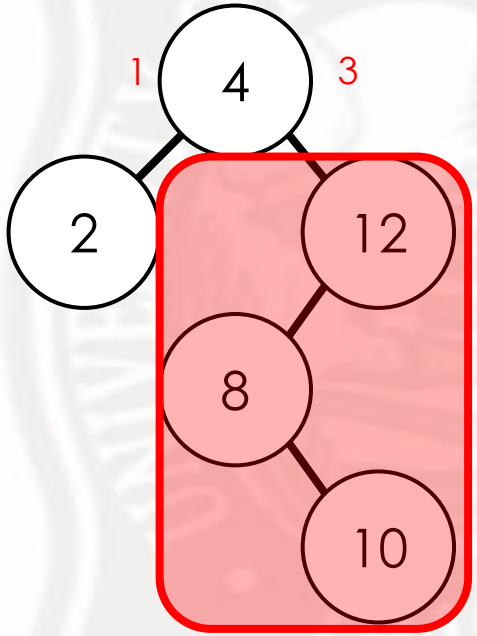




- Left subtree height: 2
- Right subtree height: 1
- 2 – 1 = /1/ = 1 ∴ an AVL tree

Left-Right Rotation

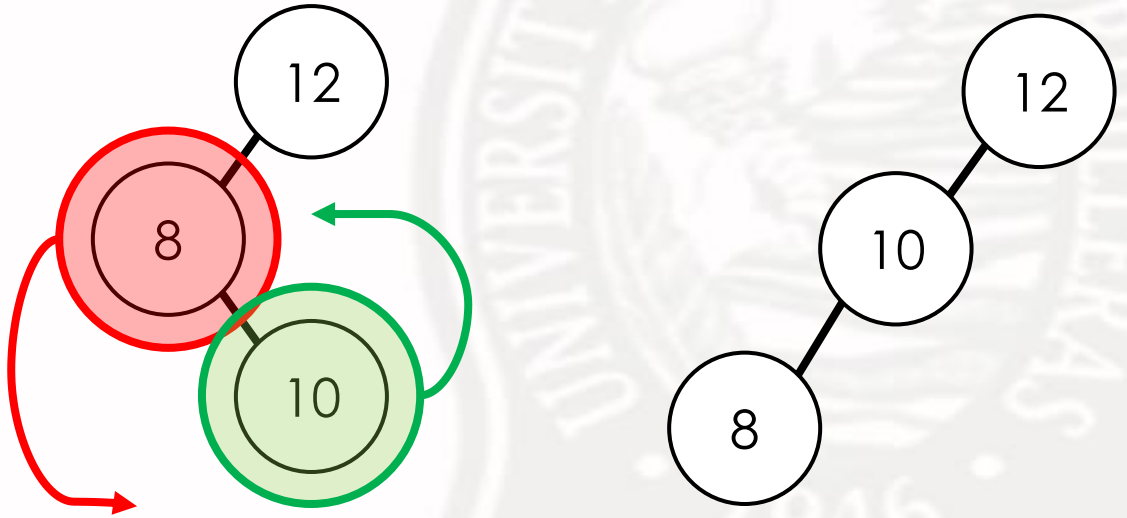
- Rotate the parent node to the left; rotate the problem node (grandchild node) to the right
- Perform right rotation
  - Rotate the grandparent node to the right, becoming the right child node of the parent node



- Left subtree height: 1
- Right subtree height: 3
- 1 – 3 = /-2/ = 2 ∴ NOT an AVL tree

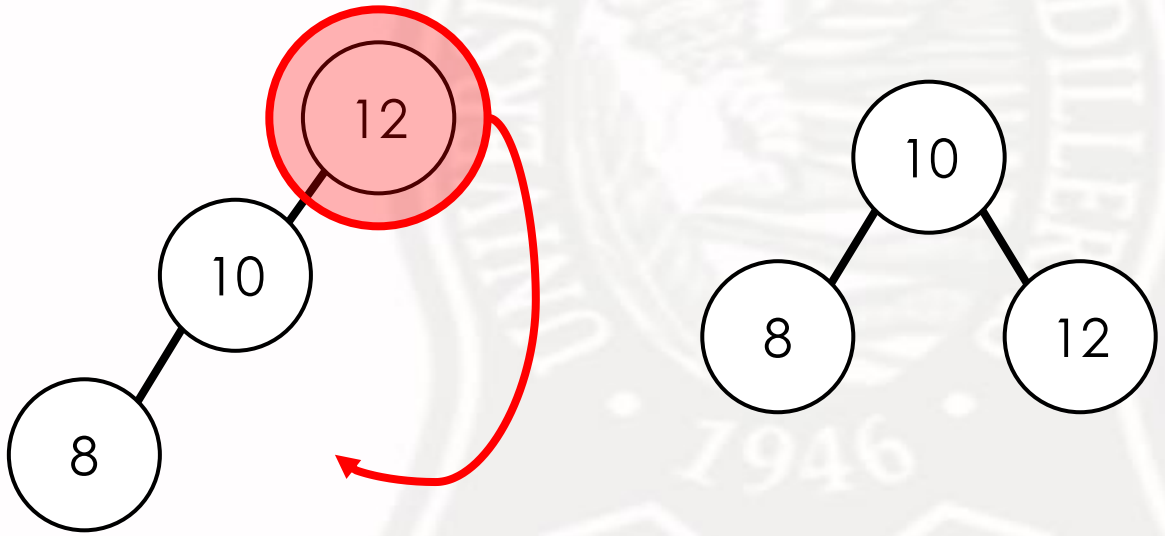
Left-Right Rotation

- **Rotate the parent node to the left; rotate the problem node (grandchild node) to the right)**
- Perform right rotation
  - Rotate the grandparent node to the right, becoming the right child node of the parent node



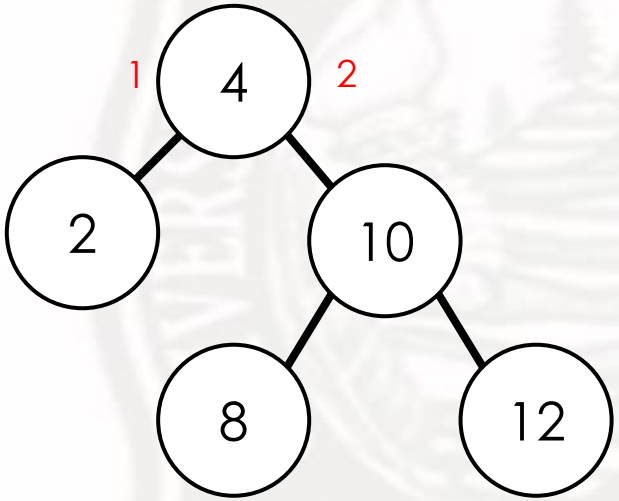
Left-Right Rotation

- Rotate the parent node to the left; rotate the problem node (grandchild node) to the right
- **Perform right rotation**
  - **Rotate the grandparent node to the right, becoming the right child node of the parent node**



Left-Right Rotation

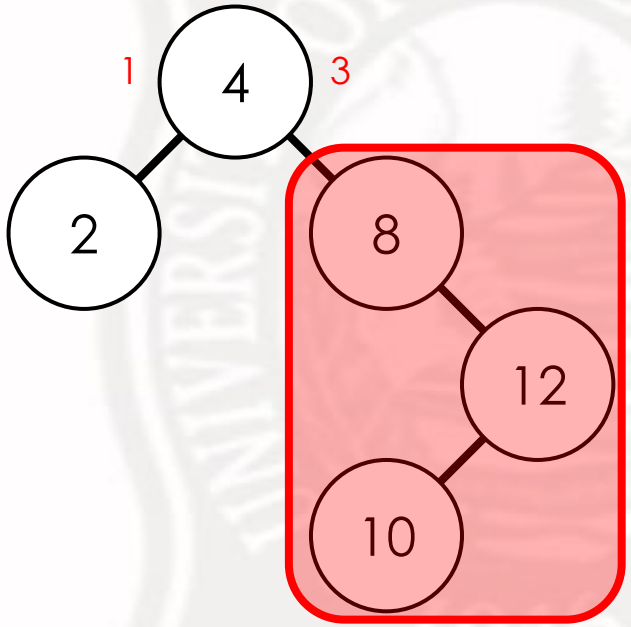
- Rotate the parent node to the left; rotate the problem node (grandchild node) to the right
- Perform right rotation
  - Rotate the grandparent node to the right, becoming the right child node of the parent node



- Left subtree height: 1
- Right subtree height: 2
- 1 – 2 = /-1/ = 1 ∴ an AVL tree

Right-Left Rotation

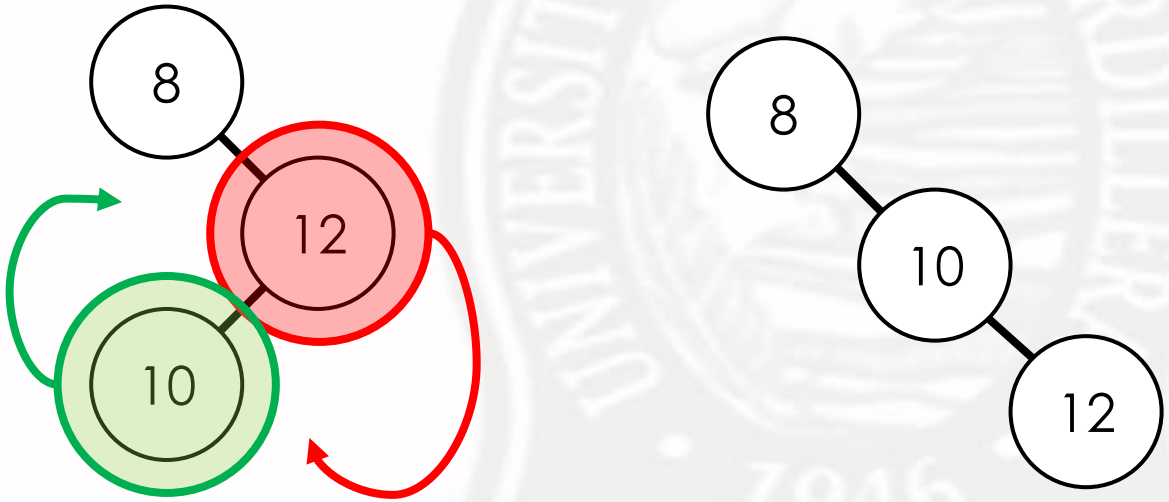
- Rotate the parent node to the right; rotate the problem node (grandchild node) to the left
- Perform left rotation
  - Rotate the grandparent node to the left, becoming the left child node of the parent node



- Left subtree height: 1
- Right subtree height: 3
- 1 – 3 = /-2/ = 2 ∴ NOT an AVL tree

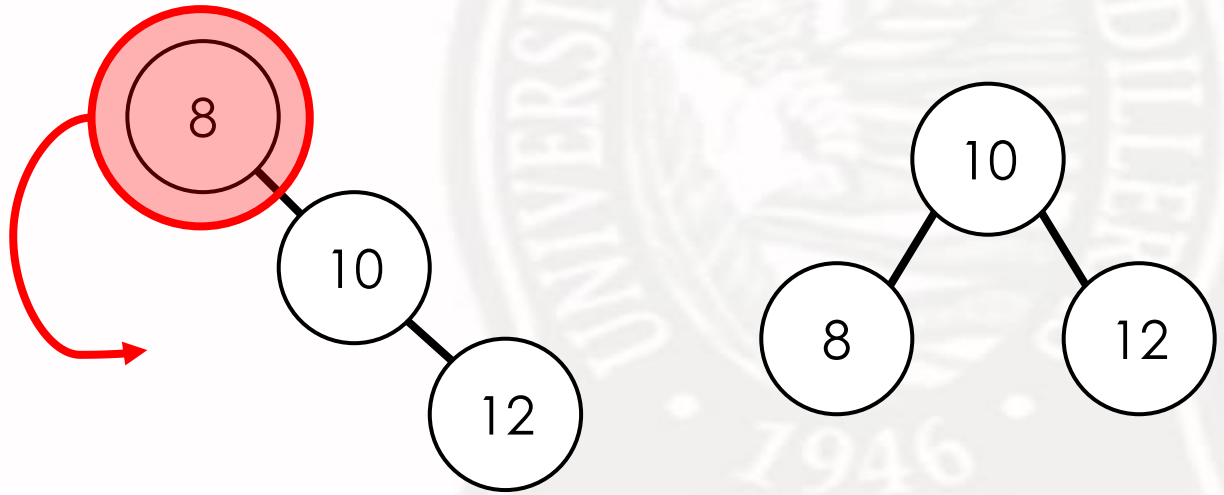
Right-Left Rotation

- **Rotate the parent node to the right; rotate the problem node (grandchild node) to the left**
- Perform left rotation
  - Rotate the grandparent node to the left, becoming the left child node of the parent node



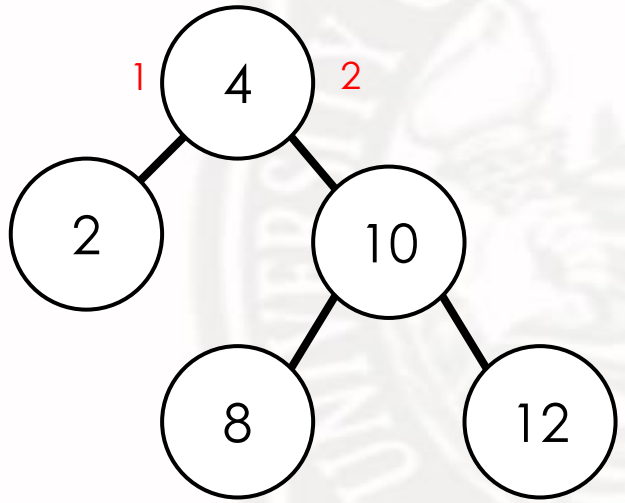
Right-Left Rotation

- Rotate the parent node to the right; rotate the problem node (grandchild node) to the left
- Perform left rotation
  - Rotate the grandparent node to the left, becoming the left child node of the parent node



Right-Left Rotation

- Rotate the parent node to the right; rotate the problem node (grandchild node) to the left
- Perform left rotation
  - Rotate the grandparent node to the left, becoming the left child node of the parent node



- Left subtree height: 1
- Right subtree height: 2
- 1 – 2 = /-1/ = 1 ∴ an AVL tree