Balanced Binary Search Tree
(AVL Tree)

Outline

This topic covers balanced binary search trees (AVL Tree):

- Background
- Definition and examples
- Implementation details of AVL operations

Background

From previous lectures:

– Binary search trees store linearly ordered data

– Best case height: $\Theta(\ln(n))$

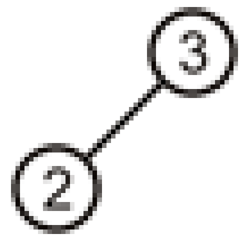
– Worst case height: $\mathbf{O}(n)$

Requirement:

– Define and maintain a *balance* to ensure $\Theta(\ln(n))$ height

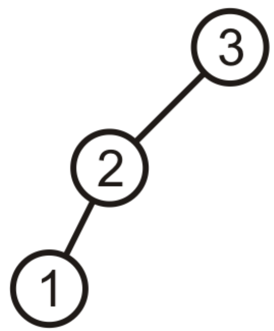
Prototypical Examples

These two examples demonstrate how we can correct for imbalances: starting with this tree, add 1:



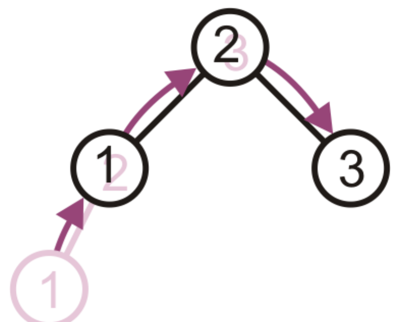
Prototypical Examples

This is more like a linked list; however, we can fix this…



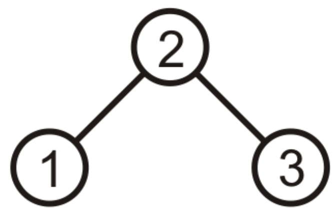
Prototypical Examples

Promote 2 to the root, demote 3 to be 2’s right child, and 1 remains the left child of 2



Prototypical Examples

The result is a perfect, though trivial tree



AVL Trees

We will focus on the: AVL trees

- Named after Adelson-Velskii and Landis

Balance is defined by comparing the height of the two sub-trees

Recall:

- An empty tree has height -1
- A tree with a single node has height 0

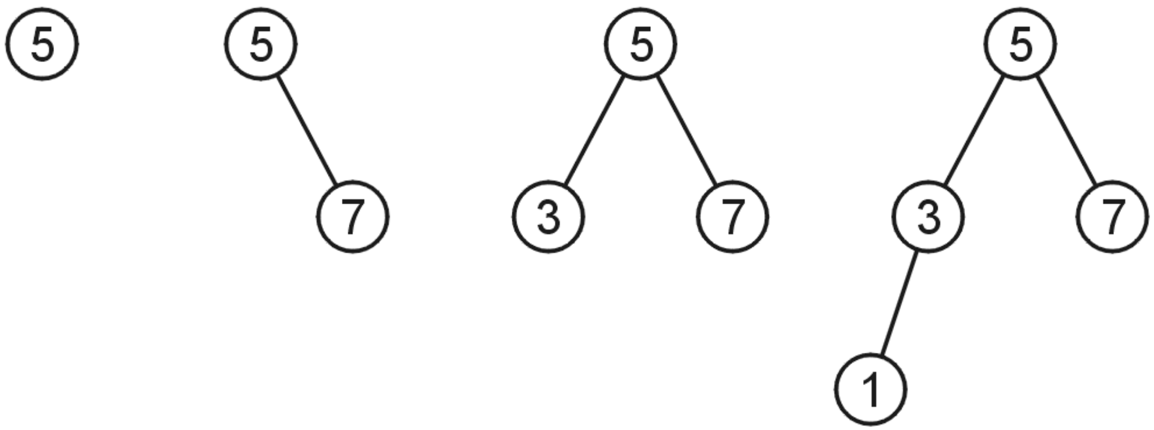
AVL Trees

A binary search tree is said to be AVL balanced if:

- The difference in the heights between the left and right sub-trees is at most 1, and
- Both sub-trees are themselves AVL trees

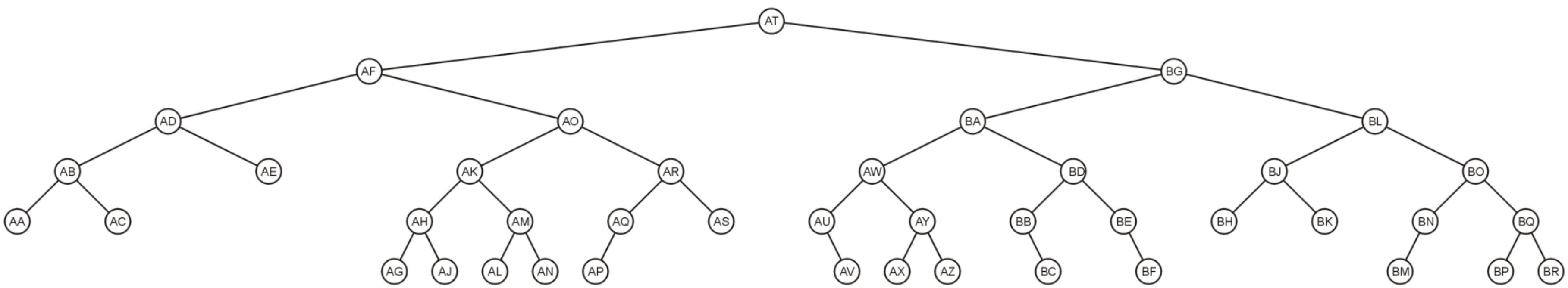
AVL Trees

AVL trees with 1, 2, 3, and 4 nodes:



AVL Trees

Here is a larger AVL tree (42 nodes):



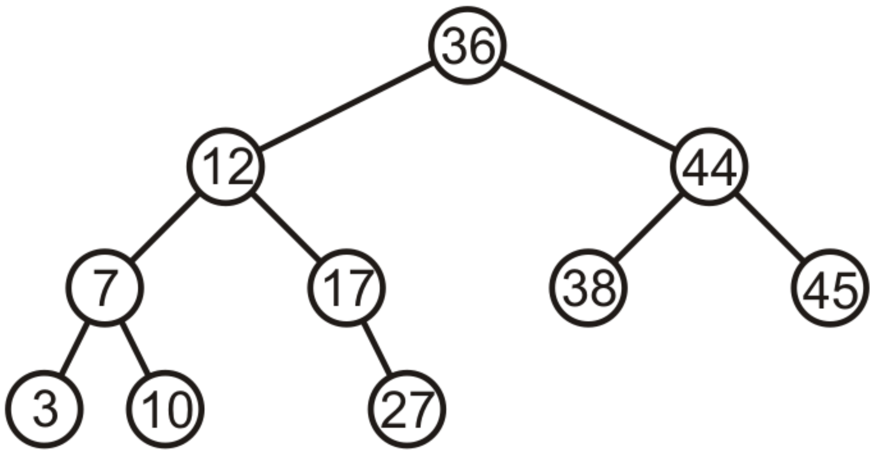
Maintaining Balance

To maintain AVL balance, observe that:

- Inserting a node can increase the height of a tree by at most 1
- Removing a node can decrease the height of a tree by at most 1

Maintaining Balance

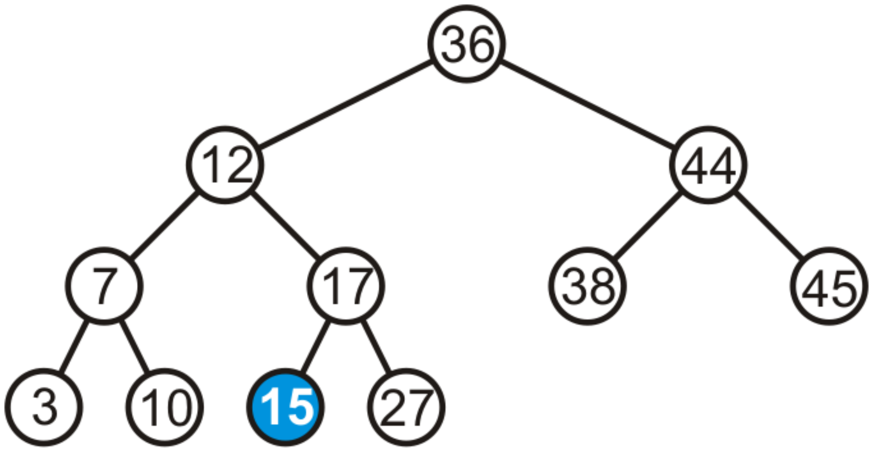
Consider this AVL tree



Maintaining Balance

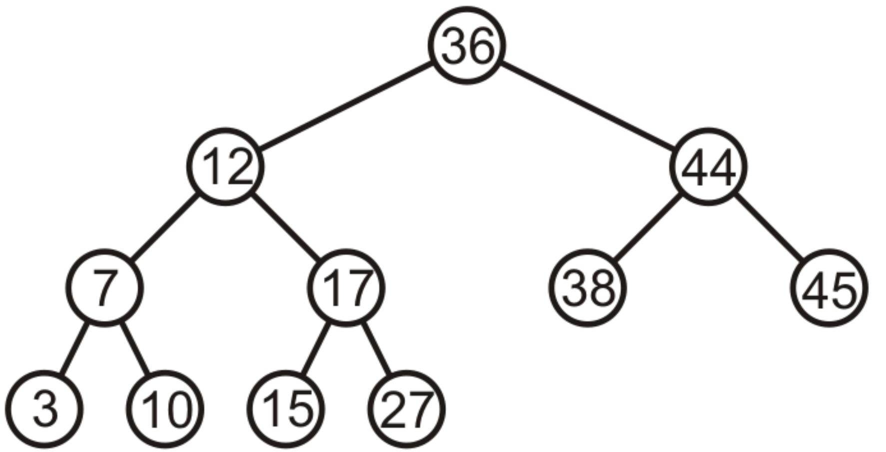
Consider inserting 15 into this tree

- In this case, the heights of none of the trees change



Maintaining Balance

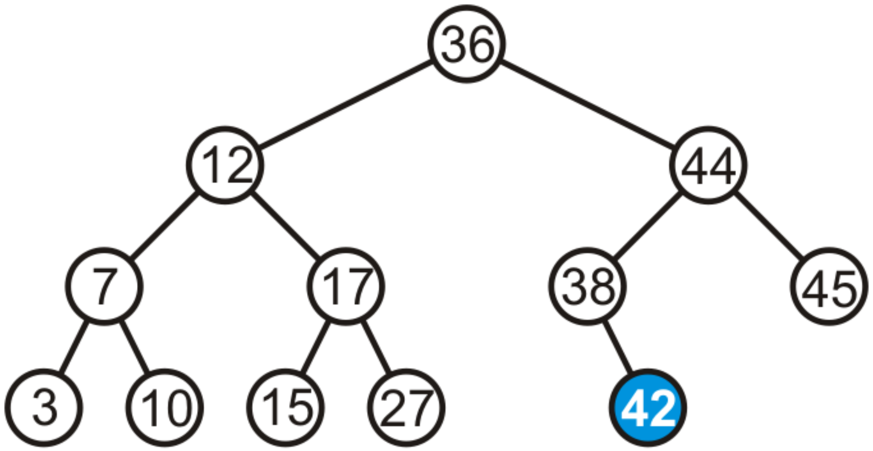
The tree remains balanced



Maintaining Balance

Consider inserting 42 into this tree

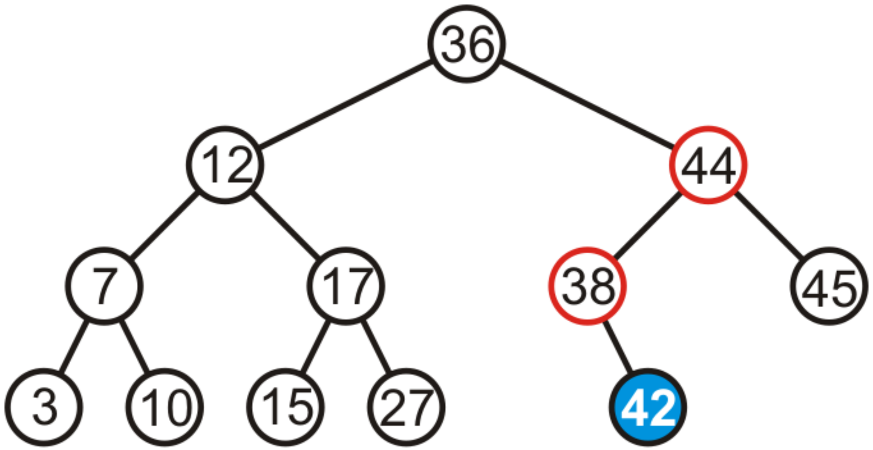
- In this case, the heights of none of the trees change



Maintaining Balance

Consider inserting 42 into this tree

- Now we see the heights of two sub-trees have increased by one
- The tree is still balanced



Maintaining Balance

Only insert and erase may change the height

- This is the only place we need to update the height
- These algorithms are already recursive

Types of Imbalance

- Left-Left Imbalance
- Right-Right Imbalance
- Left-Right Imbalance
- Right-Left Imbalance

Rotations for Maintaining Balance

- Left-Left Imbalance perform Right Rotation
- Right-Right Imbalance perform Left Rotation
- Left-Right Imbalance
  - First perform Left Rotation on left->right
  - Then perform Right Rotation
- Right-left Rotation
  - First perform Right Rotation on right->left
  - Then perform Left Rotation

AVL Tree Operations

- Examples and Code discussed in class

Summary

In this topic we have covered:

- AVL balance is defined by ensuring the difference in heights is 0 or 1
- Insertions and erases are like binary search trees
- Each insertion requires at least one correction to maintain AVL balance
- Erases may require $\mathrm{O}(h)$ corrections
- These corrections require $\Theta(1)$ time
- Depth is $\boldsymbol{\Theta}(\ln(n))$

$\therefore$ all $\mathbf{O}(h)$ operations are $\mathbf{O}(\ln(n))$