AVL Trees
Goals

- Pros/Cons of a BST
- AVL Solution
  - Height-Balanced Trees
Binary Search Tree: Balance
Binary Search Tree: Balance
Complete Binary Trees

- Very costly to maintain a complete binary tree

Diagram:
- Alex
  - Abner
    - Abigail
    - Adam
  - Angela
    - Adela
    - Alice

Add Adam to tree
• For each node, the **height difference** between the left and right subtrees is **at most one**

• Trees are locally balanced, but globally they can be slightly more unbalanced

• Null child height = -1
• Are these trees height balanced? If not, which node is out of balance?
• Are these trees height balanced? If not, which node is out of balance?
Height-Balanced Trees

• Mathematically, the longest path in a height-balanced tree has been shown to be, at worst, 44% longer than $\log n$

• Therefore, algorithms on height-balanced trees that run in time proportional to the path length are still $O(\log n)$

• So.....How do we maintain height balance??
AVL Trees

- Named after the inventors’ initials: G.M. Adelson-Velskii, E.M. Landis
- Maintain the height balanced property of a BST through a series of rotations
- When unbalanced, performs a “rotation” to balance the tree
Two Cases to Remember (Mirror for Right)

1. **Heavy left child becomes heavy itself on the left!**
   - (Single Rot)

2. **Heavy left child becomes heavy itself on the right!**
   - (Double Rot)

After operation:
- **1** is the root node.
- **2** is a child of **1**.

Diagram:
- Node 2 is unbalanced.
- After operation, the tree structure is corrected.

Illustration:
- Before and after the operations are depicted with arrows indicating the direction of the rotation.
Fix with Rotations...
Case 1: Single Rotation

Unbalanced "top" node

1(2)

2(1)

3(0)

Rotate left

2(1)

1(0)

3(0)
Case 1: Single Rotation Example 2

Unbalanced "top" node

New "top" node

Rotate left
Pseudocode to rotate current ("top") node left:

1. New top node is the current top node’s right child
2. Current node’s new right child is the new top node’s left child
3. New top’s left child is the current node
4. Set height of current node
5. Set height of new top node
6. Return new top node
Case 2: Double Rotation

- Sometimes a single rotation will not fix the problem:
  - Can happen when an insertion is made on the left (or right) side of a node that is itself a heavy right (or left) child
Case 2: Double Rotation

- Fortunately, this case is easily handled by rotating the child before the regular rotation:
  1. First rotate the heavy right (or left) child to the right (or left)
  2. Rotate the unbalanced node to the left (or right)
Case 1: Single Rotation (another view)

- Have a node (2) that is heavy on the left (1)
- Operation makes (2) unbalanced and that heavy child heavy on left
Case 1: Single Rotation

- Single rotation fixes the problem
Case 2: Double Rotation

- Have a node (3) with a heavy left child (1)
- Operation makes (3) unbalanced and (1) is heavy on the right
Case 2: Double Rotation

- First, rotate heavy child (1) to the left
Case 2: Double Rotation

- Next, rotate unbalanced node (3) to the right.

Before Rotation:

```
1
 /   
2     3
 /     
1      
```

After Rotation 2:

```
1
 /   
2     3
 /     
1      
```
Balancing pseudocode (to rebalance an unbalanced node):

If left child is tallest (by more than 1):
    If left child is heavy on the right side:  // Double rotation needed.
        Rotate the left child to the left
    Rotate unbalanced “top” node to the right
Else if right child is tallest (by more than 1)
    If right child is heavy on the left side:  // Double rotation needed.
        Rotate the right child to the right
    Rotate unbalanced “top” node to the left
Return new “top” node
Rotations

• Can be the result of additions or removals
• All cases hold for the “right” side as well, just replace all lefts with rights and all rights with lefts
More Examples...

Balanced Tree

3(3)
2(1)  8(2)
1(0)  5(1)  9(0)
4(0)  6(0)

Add data: 7

Unbalanced Tree

3(4)
2(1)  8(3)
1(0)  5(2)  9(0)
4(0)  6(1)
7(0)

“Heavy” left child

Added to right side of heavy left child

Unbalanced “top” node
AVL Trees: Double Rotation Example

Unbalanced Tree

Single rotation

Tree Still Unbalanced

Unbalanced "top" node (still)
AVL Trees: Double Rotation Example

Unbalanced Tree

- Node 3 (4)
- Node 2 (1)
- Node 8 (3)
- Node 1 (0)
- Node 5 (2)
- Node 9 (0)
- Node 4 (0)
- Node 6 (1)
- Node 7 (0)

Tree Still Unbalanced, but ...

- Node 3 (4)
- Node 2 (1)
- Node 8 (3)
- Node 1 (0)
- Node 6 (2)
- Node 9 (0)
- Node 5 (1)
- Node 7 (0)
- Node 4 (0)

Rotate heavy child

“Heavy” left child
AVL Trees: Double Rotation Example

Unbalanced Tree (after 1\textsuperscript{st} rotation)

- 3(4)
  - 2(1)
    - 1(0)
    - 5(1)
      - 4(0)
  - 8(3)
    - 9(0)
    - 7(0)

Tree Now Balanced

- 3(3)
  - 2(1)
    - 1(0)
    - 4(0)
  - 6(2)
    - 5(1)
    - 7(0)
    - 8(1)
    - 9(0)
Your Turn

• Worksheet: AVL Practice