CS 310: Tree Rotations and AVL Trees

Chris Kauffman

Week 12-2
Practice/Demo Sites

- jGrasp is so-so for seeing tree operations
- Play with Balanced Binary Search Trees online using the following applets (titles hyperlinked)

Arsen Gogeshvili: Tree Applet
- Requires Flash
- Standard BSTs
  - Manual Rotation
  - Great Practice
- AVL Trees
- Undo/Redo to rewatch
- Step by step logging

Adjustable Demo
- Standard BSTs
- All three Balanced
  - AVL, Red black, Splay
- Slow down, pause, show balance factors

Scaling AVL  (broken)
- AVL Tree only
- Scaling view for large trees
Why Worry About Insertion and Removal?

- **Q:** Why worry about insert/remove messing with the tree? What affect can it have on the performance of future ops on the tree?
- **Q:** What property of a tree dictates the runtime complexity of its operations?
Balancing Trees

- add/remove/find complexity $O(\text{height}(t))$
- Degenerate tree has height $N$: a linked list
- Prevent this by re-balancing on insert/remove
- Several kinds of trees do this
  - **AVL** left/right subtree height differ by max 1
  - **Red-black** preserve 4 red/black node properties
  - **AA** red-black tree + all left nodes black
  - **Splay** amortized bound on ops, very different
The AVL tree is named after its two Soviet inventors, Georgy Adelson-Velsky and E. M. Landis, who published it in their 1962 paper "An algorithm for the organization of information".
– Wikip: AVL Tree

- A self-balancing tree
- Operations
- Proof of logarithmic height
AVL Balance Property

T is an AVL tree if and only if

- $T\.left$ and $T\.right$ differ in height by at most 1
- AND $T\.left$ and $T\.right$ are AVL trees
Answers

T is an AVL tree if and only if
- T.left and T.right differ in height by at most 1
- AND T.left and T.right are AVL trees

1 Not AVL

2 AVL

3 Not AVL

4 AVL

5 Not AVL

6 AVL

7 Not AVL

8 AVL

80 not AVL

Left 0, Right 1

Left 2, Right 0

Left 2, Right 4

95 not AVL
Nodes and Balancing in AVL Trees

Track Balance Factor of trees

- balance = \[ \text{height}(t.\text{left}) - \text{height}(t.\text{right}) \]
- Must be -1, 0, or +1 for AVL
- If -2 or +2, must fix

Don't explicitly calculate height

- Adjust balance factor on insert/delete
- Recurse down to add/remove node
- Unwind recursion up to adjust balance of ancestors
- When unbalanced, rotate to adjust heights
- Single or Double rotation can always adjust heights by 1
Re-balancing usually involves

- Drill down during insert/remove
- Follow path back up to make adjustments
- Adjustments even out height of subtrees
- Adjustments are usually rotations
- Rotation changes structure of tree without affecting ordering
Single Rotation Basics

Right Rotation
Rotation node becomes the right subtree

Left Rotation
Rotation node becomes the left subtree
Fixing an Insertion with a Single Rotation

Insert 1, perform rotation to balance heights

- Right rotation at 8
Single Rotation Practice

Problem 1
- 40 was just inserted
- Rebalance tree rooted at 16
- Left-rotate 16

Problem 2
- 85 is being removed
- Rebalance tree rooted at 57
  - Right rotate 57
Question: Can this be fixed with single rotation?

56 was just inserted: restore AVL property with a single rotation?
Single Rotations Aren’t Enough

Cannot fix following class of situations with a single rotation

(a) Before rotation
(b) After rotation
Double Rotation Overview

**Left-Right**
- Left Rotate at $k_1$
- Right-rotate at $k_3$

**Right-Left**
- Right Rotate at $k_3$
- Left Rotate at $k_2$
Fixing an Insertion with a Double Rotation

Insert 5, perform two rotations to balance heights

- Problem is at 8: left height 3, right height 1
- Left rotate 4 (height imbalance remains)
- Right rotate 8 (height imbalance fixed)
Double Rotation Practice

- 35 was just inserted
- Rebalance the tree rooted at 36
- Use two rotations, at 33 and 36
- 36 should move
1. What is the binary search tree property?

2. What property of BSTs dictates the complexity of \( \text{find}(x), \text{insert}(x), \text{remove}(x) \)?

3. What is the memory overhead of a BST?

4. What distinguishes an AVL tree from a BST?
   - Is every AVL tree a BST?
   - Is every BST and AVL tree?

5. What kind of operation is used to maintain AVL trees during insertion/removal?
Exercise: Rebalance This AVL Tree

- Inserted 51
- Which node is unbalanced?
- Which rotation(s) required to fix?
Rebalancing Answer

Insert 51

35 Unbalanced, inserted right-left

Right rotate 57

Left rotate 35
Code for Rotations?

class Node<T>{
    Node<T> left, right;
    T data;
    int height;
}

Write the following codes for single/double rotations:

// Single Right rotation
// t becomes right child, t.left becomes new root which is returned
Node<T> rightRotate( Node<T> t ) { ... }

// Left-Right Double Rotation:
// left-rotate t.left, then right-rotate t
Node<T> leftRightRotate( Node<T> t ){ ... }
Example Rotation Codes

// Single Right rotation
Node<T> rightRotate( Node<T> t ) {
    Node<T> newRoot = t.left;
    t.left = newRoot.right;
    newRoot.right = t;
    t.height = Math.max(t.left.height,
                        t.right.height)+1;
    newRoot.height = Math.max(newRoot.left.height,
                                newRoot.right.height)+1;
    return newRoot;
}

// Left-Right Double Rotation:
// left-rotate t.left, then right-rotate t
Node<T> leftRightRotate( Node<T> t ){
    t.left = leftRotate(t.left);
    return rightRotate(t);
}

Computational complexities of these methods?
Rotations During Insertion

- Insertion works by first recursively inserting new data as a leaf.
- Tree is "unstitched" - waiting to assign left/right branches of intermediate nodes to answers from recursive calls.
- Before returning, check height differences and perform rotations if needed.
- Allows left/right branches to change the nodes to which they point.
Double or Single Rotations?

- Insert / remove code needs to determine rotations required
- Can simplify this into 4 cases

Tree T has left/right imbalance after insert(x) / remove(x)

**Zig-Zig**

\[
\begin{align*}
    &T.\text{left} > T.\text{right}+1 \\
    &T.\text{left}.\text{left} > T.\text{left}.\text{right}
\end{align*}
\]

Single Right Rotation at T

**Zag-Zag**

\[
\begin{align*}
    &T.\text{right} > T.\text{left}+1 \\
    &T.\text{right}.\text{right} > T.\text{right}.\text{left}
\end{align*}
\]

Single Left Rotation at T

**Zig-ZAG**

\[
\begin{align*}
    &T.\text{left} > T.\text{right}+1 \\
    &T.\text{left}.\text{right} > T.\text{left}.\text{left}
\end{align*}
\]

Double Rotation: left on T.left, right on T

**Zag-Zig**

\[
\begin{align*}
    &T.\text{right} > T.\text{left}+1 \\
    &T.\text{right}.\text{left} > T.\text{right}.\text{right}
\end{align*}
\]

Double Rotation: right on T.right, left on T
Excerpt of Insertion Code

From old version of Weiss AvlTree.java, in this week’s codepack

- Identify subtree height differences to determine rotations
- Useful in removal as well

```java
private AvlNode insert( Comparable x, AvlNode t ){
    if( t == null ){ // Found the spot to insert
        t = new AvlNode( x, null, null ); // return new node with data
    }
    else if( x.compareTo( t.element ) < 0 ) { // Head left
        t.left = insert( x, t.left ); // Recursively insert
    } else{ // Head right
        t.right = insert( x, t.right ); // Recursively insert
    }
    if(height(t.left) - height(t.right) == 2){ // t.left deeper than t.right
        if(height(t.left.left) > t.left.right) { // outer tree unbalanced
            t = rightRotate( t ); // single rotation
        } else { // x went left-right:
            t = leftRightRotate( t ); // double rotation
        }
    }
    else{ ... } // Symmetric cases for t.right deeper than t.left
    return t;
}
```
Does This Accomplish our Goal?

- Runtime complexity for BSTs is $\text{find}(x)$, $\text{insert}(x)$, $\text{remove}(x)$ is $O(\text{Height})$
- **Proposition:** Maintaining the AVL Balance Property during insert/remove will yield a tree with $N$ nodes and Height $O(\log N)$
- Proving this means AVL trees have $O(\log N)$ operations
- **Prove it:** What do AVL trees have to do with rabbits?
AVL Properties Give \( \log(N) \) height

**Lemma (little theorem) (Thm 19.3 in Weiss, pg 708, adapted)**

An AVL Tree of height \( H \) has at least \( F_{H+2} - 1 \) nodes where \( F_i \) is the \( i \)th Fibonacci number.

**Definitions**

- \( F_i \): \( i \)th Fibonacci number (0,1,1,2,3,5,8,13,...)
- \( S \): size of a tree
- \( H \): height (assume roots have height 1)
- \( S_H \) is the smallest size AVL Tree with height \( H \)

**Proof by Induction: Base Cases True**

<table>
<thead>
<tr>
<th>Tree</th>
<th>height</th>
<th>Min Size</th>
<th>Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty</td>
<td>( H = 0 )</td>
<td>( S_0 )</td>
<td>( F_{(0+2)} - 1 = 1 - 1 = 0 )</td>
</tr>
<tr>
<td>root</td>
<td>( H = 1 )</td>
<td>( S_1 )</td>
<td>( F_{(1+2)} - 1 = 2 - 1 = 1 )</td>
</tr>
<tr>
<td>root+(left or right)</td>
<td>( H = 2 )</td>
<td>( S_2 )</td>
<td>( F_{(2+2)} - 1 = 3 - 1 = 2 )</td>
</tr>
</tbody>
</table>
Inductive Case Part 1

Consider an Arbitrary AVL tree $T$

- $T$ has height $H$
- $S_H$ smallest size for tree $T$
- Assume equation true for smaller trees
  - Notice: Left/Right are smaller AVL trees
  - Notice: Left/Right differ in height by at most 1
Inductive Case Part 2

- $T$ has height $H$
- **Assume** for height $h < H$, smallest size of $T$ is $S_h = F_{h+2} - 1$
- Suppose Left is 1 higher than Right
- Left Height: $h = H - 1$
- Left Size: $F_{(H-1)+2} - 1 = F_{H+1} - 1$
- Right Height: $h = H - 2$
- Right Size: $F_{(H-2)+2} - 1 = F_H - 1$

$$S_H = \text{size(Left)} + \text{size(Right)} + 1$$
$$= (F_{H+1} - 1) + (F_H - 1) + 1$$
$$= F_{H+1} + F_H - 1$$
$$= F_{H+2} - 1 \blacksquare$$
Fibonacci Growth

AVL Tree of with height \( H \) has at least \( F_{H+2} - 1 \) nodes.

- How does \( F_H \) grow wrt \( H \)?
- Exponentially: \( F_H \approx \phi^H = 1.618^H \)
- \( \phi \): The Golden Ratio
- So, \( \log(F_H) \approx H \log(\phi) \)
- Or, \( \log(N) \approx \text{height} \times \phi \)
- Or, \( \log(\text{size}) \approx \text{height} \times \text{constant} \)