CS 2230
CS II: Data structures
Meeting 24: balanced binary trees
Brandon Myers
University of Iowa
Today’s big ideas

• We want binary trees where \( \text{height} \in O(\log n) \), and we’ll achieve that with balanced trees: the heights of the children of a node can differ by at most 1

• The **AVL tree** is balanced binary tree and rebalances itself using “rotations”
class BinarySearchTree {
    Node root;

    public void insert(int x) {
        if (x > data) {
            if (right == null) {
                right = new Node(x);
            } else {
                right.insert(x);
            }
        } else if (x < data) {
            if (left == null) {
                left = new Node(x);
            } else {
                left.insert(x);
            }
        } else {
            throw new IllegalArgumentException("Duplicate");
        }
    }
}

class Node {
    int data;
    Node left;
    Node right;
}
class BinarySearchTree {
    Node root;

    public void insert(int x) {
        if (x > data) {
            if (right == null) {
                right = new Node(x);
            } else {
                right.insert(x);
            }
        } else if (x < data) {
            if (left == null) {
                left = new Node(x);
            } else {
                left.insert(x);
            }
        } else {
            throw new IllegalArgumentException("Duplicate");
        }
    }
}

class Node {
    int data;
    Node left;
    Node right;
}

Peer instruction: draw the binary tree for
insert order X: 33, 24, 18, 10, 7, 5, 2

and draw the binary tree for
insert order Y: 10, 5, 24, 2, 7, 18, 33

<table>
<thead>
<tr>
<th></th>
<th>final height for X</th>
<th>final height for Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>d</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>e</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

https://b.socrative.com/login/student/
CS2230A ids 1000-4999
CS2230B ids 5000+
Same data – two different trees

unlucky insert order: 33, 24, 18, 10, 7, 5, 2
worst case time for search = height = 7

lucky insert order: 10, 5, 24, 2, 7, 18, 33
worst case time for search = height = 3
What do we want?

Most generally, we want a binary tree where $n$ nodes
height $\in O(\log n)$ (can’t do better than that)
Recall: best possible height is $O(\log N)$

<table>
<thead>
<tr>
<th>$N$</th>
<th>height</th>
<th>bottoms</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

$N = 2^{height} + bottoms - 1$

$(N - bottoms + 1) = 2^{height}$

$\log_2(N - bottoms + 1) = height$

or equivalently

$height = \lfloor \log_2 N \rfloor$

and, we know

$\lfloor \log_2 N \rfloor \in O(\log N)$
What is balanced?

Most generally, we want a binary tree where n nodes $\text{height} \in O(\log n)$ (can’t do better than that)

We’ll define a balanced binary tree as one where the above is true.

**AVL tree** is a binary tree data structure that is balanced because the *height/depth of all leaves differs by no more than 1*. 
An AVL Tree is a binary search tree such that for every internal node v of T, the heights of the children of v can differ by at most 1.

An example of an AVL tree where the heights are shown next to the nodes.
An AVL Tree is a binary search tree such that for every internal node $v$ of $T$, the heights of the children of $v$ can differ by at most 1.

Theorem: the height of an AVL tree storing $n$ nodes is $O(\log n)$
Proof: in the textbook chapter 11.3
An AVL Tree is a binary search tree such that for every internal node $v$ of $T$, the heights of the children of $v$ can differ by at most 1.

The above property of AVL trees is an *invariant* of the data structure.
Peer instruction

An AVL Tree is a binary search tree such that for every internal node $v$ of $T$, the heights of the children of $v$ can differ by at most 1.

Put these values into an AVL tree:

6  55  3  25  87  10  9  76  96  71

Done early?
See how many unique AVL trees you can draw using these values.
An AVL Tree is a binary search tree such that for every internal node \( v \) of \( T \), the heights of the children of \( v \) can differ by at most 1.

The above property of AVL trees is an *invariant* of the data structure.

How do we make sure \( \text{insert}() \) doesn’t break the invariant?
Insertion

- Insertion is as in a binary search tree
- Always done by expanding an external node.
- Example:
Trinode Restructuring

- Let \((a, b, c)\) be the inorder listing of \(x, y, z\)
- Perform the rotations needed to make \(b\) the topmost node of the three

Single rotation around \(b\)

© 2014 Goodrich, Tamassia, Goldwasser
Trinode Restructuring

• Let \((a, b, c)\) be the inorder listing of \(x, y, z\)
• Perform the rotations needed to make \(b\) the topmost node of the three

Single rotation around \(b\)

Double rotation around \(c\) and \(a\)
we inserted 54 as we normally would
unbalanced...

second step is to rebalance by performing rotations

...balanced
Same example, step by step

Single Rotations

\[ T_0 \quad T_1 \quad T_2 \quad T_3 \]
\[ a = x \quad b = y \quad c = z \]

\[ T_0 \quad T_1 \quad T_2 \quad T_3 \]
\[ a = z \quad b = y \quad c = x \]

\[ T_3 \quad T_2 \quad T_1 \quad T_0 \]
\[ a = x \quad b = y \quad c = z \]
Peer instruction

Insert the number 39 into this tree using the two steps: insert then rebalance with rotations

If you finish early: proceed to insert the number 45
AVL tree simulation

http://www.cs.usfca.edu/~galles/visualization/AVLtree.html

(also linked from the course website Resources page)
Today’s big ideas

• We want binary trees where height ∈ \( O(\log n) \), and we’ll achieve that with balanced trees: the heights of the children of a node can differ by at most 1

• The **AVL tree** is balanced binary tree and rebalances itself using “rotations”