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

• Identify whether a binary tree satisfies the **balance property**, and if not, which node is responsible

• Given some keys, draw binary trees that satisfy the balance property

• Identify the nodes involved in a **tri-node restructuring**

• Use a tri-node to **rebalance** the tree

• Execute AVLTree insertion
In section today

• Remember to bring pre-lab 11 completed
• building a Set using a binary search tree

• your TA will pass back midterms
  • see ICON for debrief announcement (Exam Improvement, Regrades)
  • **Monday Nov 6 in Debug Your Brain** (4-5pm, B11 MLH)
    I’ll be sharing some metacognitive learning strategies to help you improve before the Final
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;
}
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
\[ \text{height} \in O(\log n) \] (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)

The balance property:
\[ \text{height of all children of each internal node differs} \]
\[ \text{by no more than 1.} \]
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).

The balance property:

height of all children of each internal node 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.
Identify whether a binary tree satisfies the balance property

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.

https://b.socrative.com/login/student/
room CS2230X ids 1000-2999
room CS2230Y ids 3000+
Given some keys, draw binary trees that satisfy the balance property.

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.

https://b.socrative.com/login/student/
room CS2230X ids 1000-2999
room CS2230Y ids 3000+
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 `insert()` doesn’t break the invariant?**
Insertion

• Insertion is as in a binary search tree
• Always done by expanding an external node.
• Example:
Insert messes up the heights

\[ h-1 \]

\[ h \]

\[ h \]

\[ h \]

\[ h+1 \]

\[ h+2 \]

\[ h \]

\[ h \]

\[ h-1 \]

\[ h-1 \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]

\[ h \]
Identify whether a binary tree satisfies the **balance property**, and if not, which node is responsible.

After the insert, which node now doesn’t satisfy the AVL balance property?
- a) a
- b) b
- c) c
- d) d
- d) multiple nodes

[Diagram of binary tree with labels and insert(d) arrow]

[Link to Socrative Student Room: https://b.socrative.com/login/student/]
room CS2230X ids 1000-2999
room CS2230Y ids 3000+
To get the answer

To illustrate, consider the following scenario. Initially, we have a balanced binary tree structure with levels labeled $T_0, T_1, T_2, T_3$. The heights of these levels are denoted by $h, h+1, h-1, h-1$ respectively.

When the function `insert(d)` is called, the tree structure changes as follows:

1. The height of level $T_0$ increases from $h$ to $h+1$.
2. The height of level $T_1$ remains unchanged at $h$.
3. The heights of levels $T_2$ and $T_3$ remain at $h-1$.

The updated tree structure reflects the new height $h+2$ for level $T_0$, while the rest of the tree maintains its height as indicated.

This process demonstrates how the tree dynamically adjusts to accommodate new elements while preserving its balanced structure.
Tri-node restructuring

Goal of tri-node restructuring: rearrange 3 roots to rebalance the tree

© 2014 Goodrich, Tamassia, Goldwasser
Tri-node restructuring

Intuition: we want to **make the middle node** (here, it is b) **the new root** of the subtree

How do we pick the 3 nodes in the restructuring? Root is the node that now has unbalanced children (here, a), then include two nodes below it on the path to d (here, b and c)
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 \(b\)
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

\[
\begin{align*}
T_0 & \quad T_1 \quad T_2 \quad T_3 \\
\text{Single rotation} \quad & \quad \text{Double rotation around c} \\
\text{around b} \quad & \quad \text{and a}
\end{align*}
\]
we inserted 54 as we normally would be unbalanced...

second step is to rebalance by performing rotations

...balanced
Identify the nodes involved in a tri-node restructuring

Node 54 was just inserted, what three nodes should be included in the tri-node restructuring?

https://b.socrative.com/login/student/
room CS2230X ids 1000-2999
room CS2230Y ids 3000+
Answer
Another example

\[
\begin{align*}
\text{a} &= z \\
\text{b} &= x \\
\text{c} &= y \\
T_0 & & T_1 & & T_2 & & T_3 \\
T_0 & & T_1 & & T_2 & & T_3
\end{align*}
\]

insert(d)
We might need to rebalance the tree. What will be the result?
a) no change
b) a remains the root but left=b
c) b is the new root with left=a, right=c
d) c is the new root with left=a, right=b
e) d is the new root with left=b, right=c
We might need to rebalance the tree. What will be the result?

a) no change
b) a remains the root but left=b
c) b is the new root with left=a, right=c
d) c is the new root with left=a, right=b
e) d is the new root with left=b, right=c
We might need to rebalance the tree. What will be the result?

a) no change
b) a remains the root but left=b
c) b is the new root with left=a, right=c
d) c is the new root with left=a, right=b
e) d is the new root with left=b, right=c
Implementing tri-node restructuring

• Step 1: traverse up from the inserted node, checking the AVL balance property at each node. Once you find once, you have your 3 nodes involved in the tri-node.

check d,
check its parents...
check b,
check c,
check a (aha! imbalanced)
Implementing tri-node restructuring

- Step 1: traverse up from the inserted node, checking the AVL balance property at each node. Once you find once, you have your 3 nodes involved in the tri-node

- Step 2: do either 1 rotation or 2 rotations depending on which of four cases...
Implementing tri-node restructuring

Single rotations

Case 1: right, right

Case 2: left, left

© 2014 Goodrich, Tamassia, Goldwasser
Implementing tri-node restructuring

Double rotations

- Case 3: right, left

- Case 4: left, right
AVL tree simulation

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

(also linked from the course website Resources page)
Recall the tree where we identified the Tri-node.

What rotations are needed?
Which case is needed to perform either 1 or 2 rotations?

a) case 1: right, right
b) case 2: left, left
c) case 3: right, left
d) case 4: left, right
Answer

left rotate (50)

right rotate (78)
Execute AVLTree insertion

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
after an insert imbalances a AVLTree of height H, how many tri-node restructurings do we have to perform to balance the whole tree? (Answer in terms of N, the number of nodes in the tree)

a) O(1)
b) O(log N)
c) O(sqrt(N))
d) O(N log N)
e) O(N^2)
f) O(N^3)
Today’s learning objectives

• Identify whether a binary tree satisfies the **balance property**, and if not, which node is responsible

• Given some keys, draw binary trees that satisfy the balance property

• Identify the nodes involved in a **tri-node restructuring**

• Use a tri-node to **rebalance** the tree

• Execute AVLTree insertion