AVL Trees

Autumn 2018

Shrirang (Shri) Mare
shri@cs.washington.edu

Thanks to Kasey Champion, Ben Jones, Adam Blank, Michael Lee, Evan McCarty, Robbie Weber, Whitaker Brand, Zora Fung, Stuart Reges, Justin Hsia, Ruth Anderson, and many others for sample slides and materials ...
So far
- List
- Dictionaries
- Add and remove operations on dictionaries implemented with arrays or lists are $O(n)$
- Trees, BSTs in particular, offer speed up because of their branching factors
- BSTs are in the average case, but not in the worse case

<table>
<thead>
<tr>
<th></th>
<th>Insert()</th>
<th>Find()</th>
<th>Remove()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average case</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Worst case</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

Today
- Can we do better? Can we adapt our BST so we never get the worst case
Review: Worksheets
Balanced BST observation

BST: the shallower the better!

For a BST with \( n \) nodes inserted in arbitrary order

– Average height is \( \Theta(\log n) \)
– Worst case height is \( \Theta(n) \)

**Solution**: Require a **Balance Condition** that

Simple cases such as inserting in key order lead to the worst-case scenario

1. ensures depth is always \( \Theta(\log n) \) – strong enough!
2. is easy to maintain – not strong enough!
AVL trees: Balanced BSTs

**AVL Trees** must satisfy the following properties:
- **binary trees**: every node must have between 0 and 2 children
- **binary search tree (BST property)**: for every node, all keys in the left subtree must be smaller and all keys in the right subtree must be larger than the root node
- **Balanced (AVL property)**: for every node, there can be no more than a difference of 1 in the height of the left subtree from the right. \(\text{Math.abs(height(left subtree) - height(right subtree))} \leq 1\)

AVL stands for Adelson-Velsky and Landis (the inventors of the data structure)

The AVL property:

1. ensures depth is always \(\mathcal{O}(\log n)\) – Yes!
2. is easy to maintain – Yes! (using single and double rotations)
Potential balance conditions (1)

1. Left and right subtrees of the *root* have equal number of nodes

2. Left and right subtrees of the *root* have equal *height*
Potential balance conditions (1)

1. Left and right subtrees of the root have equal number of nodes
   - Too weak!
   - Height mismatch example:

2. Left and right subtrees of the root have equal height
   - Too weak!
   - Double chain example:
Potential balance conditions (2)

3. Left and right subtrees of every node have equal number of nodes

4. Left and right subtrees of every node have equal *height*
Potential balance conditions (2)

3. Left and right subtrees of every node have equal number of nodes
   
   *Too strong!*
   
   *Only perfect trees (2^n – 1 nodes)*

4. Left and right subtrees of every node have equal *height*

   *Too strong!*

   *Only perfect trees (2^n – 1 nodes)*
AVL balance condition

**AVL condition:** For every node, the height of its left subtree and right subtree differ by at most 1.

\[
\text{balance}(\text{node}) = \text{Math.abs}(\text{height(}\text{node.left}) - \text{height(}\text{node.right})
\]

**AVL condition:** for every node, \(\text{balance}(\text{node}) \leq 1\)
Worksheet (Q9)
Insertion

What happens if when we do an insert(3), we break the AVL condition?
Left Rotation

Rest of the tree

Rest of the tree

UNBALANCED
Right subtree is 2 longer

BALANCED
Right subtree is 1 longer

A
B
C
D

A
B
C
D

CSE 332 SU 18 – ROBBIE WEBER
Tree Rotations: Right rotation

[Diagram of tree rotation]
Can’t do a left rotation
Do a “right” rotation around 3 first.

There’s a “kink” in the tree where the insertion happened.

Now do a left rotation.
Right Left Rotation

Left subtree is 1 longer

Right subtree is 2 longer

UNBALANCED

BALANCED
Right subtree is 1 longer

Rest of the tree

Rest of the tree
Four cases to consider

<table>
<thead>
<tr>
<th>Insert location</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left subtree of left child of y</td>
<td>Single right rotation</td>
</tr>
<tr>
<td>Right subtree of left child of y</td>
<td>Double (left-right) rotation</td>
</tr>
<tr>
<td>Left subtree of right child of y</td>
<td>Double (right-left) rotation</td>
</tr>
<tr>
<td>Right subtree of right child of y</td>
<td>Single left rotation</td>
</tr>
</tbody>
</table>
Four cases to consider

The “line” case

The “kink” case
AVL Example: 8, 9, 10, 12, 11
AVL Example: 8,9,10,12,11
AVL Example: 8,9,10,12,11
AVL Example: 8,9,10,12,11
AVL Example: 8, 9, 10, 12, 11
Worksheet (Q10A)
Worksheet (Q10B)
How Long Does Rebalancing Take?

Assume we store in each node the height of its subtree. How do we find an unbalanced node?

How many rotations might we have to do?
How Long Does Rebalancing Take?

Assume we store in each node the height of its subtree.

How do we find an unbalanced node?
- Just go back up the tree from where we inserted.

How many rotations might we have to do?
- Just a single or double rotation on the lowest unbalanced node.
- A rotation will cause the subtree rooted where the rotation happens to have the same height it had before insertion.
Lots of cool Self-Balancing BSTs out there!

Popular self-balancing BSTs include:

- AVL tree
- Splay tree
- 2-3 tree
- AA tree
- Red-black tree
- Scapegoat tree
- Treap

(Not covered in this class, but several are in the textbook and all of them are online!)

(From https://en.wikipedia.org/wiki/Self-balancing_binary_search_tree#Implementations)