CS206

Search Trees, AVL Trees
Binary Search Trees

- For all nodes
  - The left node is less than parent
  - The right node is greater than parent
Binary Search Trees

- Performance is directly affected by the height of tree
- All operations are $O(h)
- $h = O(n)$ worst case
- $h = O(\log n)$ best case
- Expected $O(\log n)$ if tree is “balanced”
  - balance — generally same number of nodes in left and right subtrees
Balanced Search Trees

- A variety of algorithms augment a standard BST with occasional operations to reshape, reduce height and maintain balance.
- General approach: Rotation — moves a child to be above its parent,
  - ideally $O(1)$
  - certainly $O(lgn)$
Rotation Algorithms

- **AVL trees**
  - Adelson-Velski and Landis (1962)
- Splay trees
- (2,4) trees
  - non-binary trees
- Red-Black trees
AVL Trees

• Height-balance property
  □ For every internal node, the avlHeight of the two children differ by at most 1
    □ avlHeight = max distance from null endpoint

• Any binary tree satisfying the height-balance property is an AVL tree

• A height-balanced tree has height $O(\lg n)$
  □ max height is provably $1.44*\lg(n)$
AVL Tree Example
Insertion

- Maintain with each node the avlHeight.
- On insertion, first recur down through tree to insert.
- Then as you unwind recursion, update the avlHeight of each node.
- If height changes, check the height of other child
  - if not in balance then fix
Insertion code to maintain height
(the only code today!!!)

```java
private class Node {
    Comparable<E> element;
    int avlHight;
    Node right;
    Node left;

    public Node(Comparable<E> e) {
        avlHight = 1;
        element = e;
        right = null;
        left = null;
    }
}
```
int insertUtil(node, element):
    if element==node.payload
        return -1;

    avlD=2;
    if node.payload > element:
        if node.left==null
            node.left=new Node(payload)
        else
            avlD = 1+insertUtil(node.left,element);
    else
        // same but for right

    node.avlHieght = greater of avlD and
        node.avlHeight

    return node.avlHeight
Fixing height imbalances
Rotation!!

- Two types of rotation
- Single
  - left subtree of left node causes imbalance
  - right subtree of right node causes imbalance
- Double
  - right subtree of left node causes imbalance
  - left subtree of right node causes imbalance
  - The first rotation of a double puts the tree into position for a single rotation!
AVL Animation
Double Rotation

First rotate across the point imbalance

Then do a single rotation
Single Rotation

Rotate across parent at the lowest imbalance
Deletion

• Deletion removes a node with 0 or 1 child
  • recall deletion from binary tree for node with 2 children.
• Deletion may reduce the height of parent
• Rotate to rebalance just like insertion
• Fix avlHeight
• May in case of ties, choose a single rotation.
$O(\log n)$ Rotations

- Unlike insertion where rotation of the nearest unbalanced ancestor restores the balance globally.
- On deletion, rotation of the nearest unbalanced ancestor only guarantees balance locally to the subtree.
- Worst-case requires $O(\log n)$ rotations up the tree to restore balance globally.
## Doing AVL

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>insert</code> &amp; <code>insert</code> &amp; <code>insert</code> &amp; <code>insert</code> &amp; <code>insert</code> &amp; <code>insert</code> &amp; <code>insert</code> &amp; <code>insert</code> &amp; <code>insert</code></td>
<td>100</td>
</tr>
<tr>
<td><code>insert</code></td>
<td>200</td>
</tr>
<tr>
<td><code>insert</code></td>
<td>300</td>
</tr>
<tr>
<td><code>insert</code></td>
<td>400</td>
</tr>
<tr>
<td><code>insert</code></td>
<td>500</td>
</tr>
<tr>
<td><code>insert</code></td>
<td>600</td>
</tr>
<tr>
<td><code>insert</code></td>
<td>700</td>
</tr>
<tr>
<td><code>insert</code></td>
<td>800</td>
</tr>
<tr>
<td><code>insert</code></td>
<td>900</td>
</tr>
<tr>
<td><code>insert</code></td>
<td>750</td>
</tr>
<tr>
<td><code>insert</code></td>
<td>1000</td>
</tr>
<tr>
<td><code>insert</code></td>
<td>850</td>
</tr>
<tr>
<td><code>delete</code></td>
<td>400</td>
</tr>
<tr>
<td><code>delete</code></td>
<td>300</td>
</tr>
<tr>
<td><code>delete</code></td>
<td>200</td>
</tr>
<tr>
<td><code>delete</code></td>
<td>700</td>
</tr>
<tr>
<td><code>delete</code></td>
<td>500</td>
</tr>
</tbody>
</table>
Mini-Lab
AVL tree practice

Show the BST tree and each AVL rotation (if needed) to keep a BST an AVL tree

<table>
<thead>
<tr>
<th>insert</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td>500</td>
</tr>
<tr>
<td>insert</td>
<td>750</td>
</tr>
<tr>
<td>insert</td>
<td>625</td>
</tr>
<tr>
<td>insert</td>
<td>560</td>
</tr>
<tr>
<td>insert</td>
<td>590</td>
</tr>
<tr>
<td>insert</td>
<td>400</td>
</tr>
<tr>
<td>insert</td>
<td>300</td>
</tr>
<tr>
<td>insert</td>
<td>600</td>
</tr>
<tr>
<td>insert</td>
<td>200</td>
</tr>
<tr>
<td>delete</td>
<td>560</td>
</tr>
<tr>
<td>delete</td>
<td>590</td>
</tr>
</tbody>
</table>