Final Project Proposals

Please notify me when you have resubmitted

Email or post on Campuswire if you want to schedule a meeting
Final Project Development Log

Pick a day each week to write up a summary of progress

For full credit, must have an entry each week starting this week

If project is not yet approved, just discuss the progress on proposal
reflection on lab_trees

Avg: 15/15
Exam 2 October 29th - October 31st

Exam will be combination multiple choice and coding

Covers all content up until October 20th

Sign up for registration as early as October 14th

Coding question will use new workspace setup.
Learning Objectives

- Identify four AVL rotations
- Discuss how AVL functions relate to BST functions
- Prove that the AVL tree’s height is bounded
## BST Analysis – Running Time

<table>
<thead>
<tr>
<th>Operation</th>
<th>BST Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>find</td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td></td>
</tr>
<tr>
<td>remove</td>
<td></td>
</tr>
<tr>
<td>traverse</td>
<td></td>
</tr>
</tbody>
</table>
BST Rotations

We can adjust the BST structure by performing \textit{rotations}.
Left Rotation
Left Rotation
Right Rotation

```
38

13  51

10  25

12  37
```
Complex rotations
LeftRight Rotation
LeftRight Rotation

Left rotation at 13

Right rotation at 38

Diagram showing the process of left and right rotations in a binary tree.
RightLeft Rotation

```
  10
 /  \
8   15
 /    |
11   18
    /  |
   12
```
AVL Rotations

Four kinds of rotations:
AVL Rotations

Four kinds of rotations: (L, R, LR, RL)

1. All rotations are local (subtrees are not impacted)

2. The running time of rotations are constant

3. The rotations maintain BST property

Goal:
AVL Find

_find(7)
AVL Insertion

```python
class treeNode:
    def __init__(self, key, val, left=None, right=None):
        self.key = key
        self.val = val
        self.left = left
        self.right = right
```

$insert(6.5)$
AVL Insertion

class treeNode:
    def __init__(self, key, val, left=None, right=None):
        self.key = key
        self.val = val
        self.left = left
        self.right = right

def insert(self, key, val):
    self.root = self.insert_helper(self.root, key, val)

def insert_helper(self, node, key, val):
    if node == None:
        return treeNode(key, val)
    if key < node.key:
        node.left = self.insert_helper(node.left, key, val)
    else:
        node.right = self.insert_helper(node.right, key, val)
    return rebalance(node)
Rebalancing on insert

Theorem:
If an insertion occurred in subtrees $t_3$ or $t_4$ and an imbalance was first detected at $t$, then a ____________ rotation about $t$ restores the balance of the tree.

We gauge this by noting the balance factor of $t$-right is _______. 
Rebalancing on insert

**Theorem:**
If an insertion occurred in subtrees $t_2$ or $t_3$ and an imbalance was first detected at $t$, then a __________ rotation about $t$ restores the balance of the tree.

We gauge this by noting the balance factor of $t$->right is ______.
Rebalancing on insert
AVL Remove

_remove(10)
AVL Remove

_remove(10)
AVL Remove

(remove(10)
AVL Remove

_remove(10)

R@8
AVL Remove

1) find(10)
2) find( IOP / IOS )
3) swap and remove
4) rebalance
5) recurse

_\text{remove}(10)_
AVL Remove
AVL Tree Analysis

For AVL tree of height $h$, we know:

- find runs in: __________.
- insert runs in: __________.
- remove runs in: __________.

We will argue that: $h$ is __________.