CS 225
Data Structures

March 17 – AVL Analysis
Brad Solomon
Informal Early Feedback Reminder

CS 225 All SP21: Data Structures (Evans, C)

Dashboard / My courses / CS 225 All SP21

- Announcements
- Lab Attendance
- Informal Early Feedback
- Lab Informal Early Feedback
Final Project Team Formation Survey

What is your time zone in Coordinated Universal Time (UTC)? Paste in your browser for UTC. https://www.timeanddate.com/time/map/

☐ UTC -11:00
☐ UTC -10:00 (US Hawaii)
☐ UTC -9:00 (US Alaska)
☐ UTC -8:00 (US Pacific, British Columbia, Baja)
☐ UTC -7:00 (US Mountain, Alberta, W. Mexico)
☐ UTC -6:00 (US Central, E. Mexico, Manitoba)
☐ UTC -5:00 (US Eastern, Colombia, Quebec)

Make a selection ▼

What is your gender?

Please indicate the racial/ethnic group with which you most identify:

Please check the times that you are in class, at work or practice and are busy and unavailable for group work:

(You may select entire rows or columns by clicking the column/row headers)

By default, students can retake Team Maker surveys up until it is closed (midnight of the 'End Date'). If you need to change your schedule after submitting this survey, you will be allowed to retake the survey to update your schedule.

<table>
<thead>
<tr>
<th>Make Busy</th>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
<th>Sun</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:00am</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>9:00am</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>10:00am</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>11:00am</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>12:00pm</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>1:00pm</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>2:00pm</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>3:00pm</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>4:00pm</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>5:00pm</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>6:00pm</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>7:00pm</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>8:00pm</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>9:00pm</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

If you have formed a team already, what is your team’s UUID? (Be sure to submit identical IDst)
Learning Objectives

• Review AVL trees

• Formalize code for _insert and generalize to _find and _remove

• Quantify efficiency of AVL tree operations as a factor of $h$

• Develop strategies for formalizing $h$ as a mathematical expression
AVL TreeNode

AVL is a BST that maintains balance
AVL Tree Rotations

All rotations are $O(1)$

All rotations reduce subtree height by one
Insertion into an AVL Tree

Insert (pseudo code):
1: Insert at proper place
2: Check for imbalance
3: Rotate, if necessary
4: Update height

```
struct TreeNode {
    T key;
    unsigned height;
    TreeNode *left;
    TreeNode *right;
};
```
Insertion into an AVL Tree

```cpp
struct TreeNode {
    T key;
    unsigned height;
    TreeNode *left;
    TreeNode *right;
};

template <typename K, typename V>
void AVL<K, D>::_insert(const K & key, const V & data, TreeNode * & cur) {
    if (cur == NULL) { cur = new TreeNode(key, data); }
    else if (key < cur->key) { _insert(key, data, cur->left); }
    else if (key > cur->key) { _insert(key, data, cur->right); }
    _ensureBalance(cur);
}
```

**amethyst_cat2**: can we call `ensureBalance` multiple times for one insert?

![AVL Tree Visualization]
template <class T> void AVLTree<T>::_insert(const T & x, treeNode<T> * & t ) {
if( t == NULL ) {
    t = new TreeNode<T>( x, 0, NULL, NULL);
}
else if( x < t->key ) {
    _insert( x, t->left );
    int balance = height(t->right) - height(t->left);
    int leftBalance = height(t->left->right) - height(t->left->left);
    if ( balance == -2 ) {
        if ( leftBalance == -1 ) { rotate_____________( t ); }
    } 
    else
    { rotate_____________( t ); }
}
else if( x > t->key ) {
    _insert( x, t->right );
    int balance = height(t->right) - height(t->left);
    int rightBalance = height(t->right->right) - height(t->right->left);
    if( balance == 2 ) {
        if( rightBalance == 1 ) { rotate_____________( t ); }
    } 
    else
    { rotate_____________( t ); }
}
t->height = 1 + max(height(t->left), height(t->right));
}
template <class T> void AVLTree<T>::_insert(const T & x, treeNode<T> * & t) {
    if (t == NULL) {
        t = new TreeNode<T>(x, 0, NULL, NULL);
    }
    else if (x < t->key) {
        _insert(x, t->left);
        int balance = height(t->right) - height(t->left);
        int leftBalance = height(t->left->right) - height(t->left->left);
        if (balance == -2) {
            if (leftBalance == -1) { rotateRight(t); }
            else { rotateLeftRight(t); }
        }
    }
    else if (x > t->key) {
        _insert(x, t->right);
        int balance = height(t->right) - height(t->left);
        int rightBalance = height(t->right->right) - height(t->right->left);
        if (balance == 2) {
            if (rightBalance == 1) { rotateLeft(t); }
            else { rotateRightLeft(t); }
        }
    }
    t->height = 1 + max(height(t->left), height(t->right));
}
Find in an AVL Tree

```cpp
struct TreeNode {
    T key;
    unsigned height;
    TreeNode *left;
    TreeNode *right;
};
```

```cpp
template<typename K, typename V>
TreeNode *find(TreeNode *root, const K &key) const {
    if (root == nullptr) return root;
    if (root->key == key) return root;
    if (root->key > key) return find(root->left, key);
    else return find(root->right, key);
}
```
Remove from an AVL Tree

_remove(10)
Remove from an AVL Tree

_remove(10)

1) _find(10)
2) _find( __IOP / IOS__ )
3) swap and remove
Remove from an AVL Tree

_remove(10)

1) _find(10)
2) _find( __IOP / IOS__ )
3) swap and remove
4) rebalance
Remove from an AVL Tree

_\text{remove}(10)_

1) \_\text{find}(10)
2) \_\text{find}( \underline{\_\text{IOP} / \text{IOS}}) \\
3) swap and remove \\
4) rebalance \\
5) recurse
Remove from an AVL Tree

1) \_find(10)
2) \_find( __IOP / IOS__ )
3) swap and remove
4) rebalance
5) recurse

\_remove(10)
Remove from an AVL Tree

5

X
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 __________.
AVL Tree Analysis

Definition of big-O:

\[ f(n) \text{ is } O(g(n)) \text{ iff } \exists c, k \text{ s.t. } f(n) \leq cg(n) \quad \forall n > k \]

...or, with pictures:
The height of the tree, $f(n)$, will always be less than $c \times g(n)$ for all values where $n > k$. 
AVL Tree Analysis

The number of nodes in the tree, $f^{-1}(h)$, will always be greater than $c \times g^{-1}(h)$ for all values where $n > k$. 

$f(n) = “Tree\ height\ given\ nodes”$ 

$f^{-1}(h) = “Nodes\ in\ tree\ given\ height”$
Plan of Action

Since our goal is to find the lower bound on $n$ given $h$, we can begin by defining a function given $h$ which describes the smallest number of nodes in an AVL tree of height $h$:

$$N(h) = \text{minimum number of nodes in an AVL tree of height } h$$
Simplify the Recurrence

\[ N(h) = 1 + N(h - 1) + N(h - 2) \]

\[ N(h) \geq N(h) - 1 \]
State a Theorem

**Theorem:** An AVL tree of height \( h \) has at least \__________\.

**Proof by Induction:**
I. Consider an AVL tree and let \( h \) denote its height.

II. Base Case: \______________\.

An AVL tree of height \______\ has at least \______\ nodes.
Prove a Theorem

III. Base Case: ________________

An AVL tree of height ____ has at least ____ nodes.
Prove a Theorem

IV. Induction Case: ________________

If for all heights \( i < h \), \( N(i) \geq 2^{i/2} \)

then we must show for height \( h \) that \( N(h) \geq 2^{h/2} \)
Prove a Theorem

V. Using a proof by induction, we have shown that:

...and inverting: