CS 225
Data Structures

March 19 – AVL Applications
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
Learning Objectives

• Review Big O in the contexts of an AVL tree

• Formalize relationship between $n$ and $h$ in an AVL tree

• Prove $h$ has an upper bound of $O(\log n)$

• Wrap up balanced binary trees
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 __________.
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

\[ f(n) = \text{“Tree height given nodes”} \]

\[ f^{-1}(h) = \text{“Nodes in tree given height”} \]

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

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

II. Base Case: \( \underline{\text{______________}} \)

An AVL tree of height \( \underline{\text{_____}} \) has at least \( \underline{\text{_____}} \) 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:
AVL Runtime Proof

An upper-bound on the height of an AVL tree is $O(\lg(n))$:

$$N(h) := \text{Minimum # of nodes in an AVL tree of height } h$$
$$N(h) = 1 + N(h-1) + N(h-2)$$
$$> 1 + 2^{h-1/2} + 2^{h-2/2}$$
$$> 2 \times 2^{h-2/2} = 2^{h-2/2+1} = 2^{h/2}$$

**Theorem #1:**

Every AVL tree of height $h$ has at least $2^{h/2}$ nodes.
AVL Runtime Proof

An upper-bound on the height of an AVL tree is $O(\lg(n))$:

$$\# \text{ of nodes } (n) \geq N(h) > 2^{h/2}$$
$$n > 2^{h/2}$$
$$\lg(n) > h/2$$
$$2 \times \lg(n) > h$$
$$h < 2 \times \lg(n), \text{ for } h \geq 1$$

Proved: The maximum number of nodes in an AVL tree of height $h$ is less than $2 \times \lg(n)$. 
Summary of Balanced BST

**AVL Trees**
- Max height: $1.44 \times \lg(n)$
- Rotations:
Summary of Balanced BST

**AVL Trees**
- Max height: 1.44 * \( \lg(n) \)
- Rotations:
  Zero rotations on find
  One rotation on insert
  \( O(h) = O(\lg(n)) \) rotations on remove

**Red-Black Trees**
- Max height: 2 * \( \lg(n) \)
- Constant number of rotations on insert (max 2), remove (max 3).
Red-Black Trees in C++

C++ provides us a balanced BST as part of the standard library:

```cpp
std::map<K, V> map;
```

### Lookup

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear</td>
<td>clears the contents (public member function)</td>
</tr>
<tr>
<td>insert</td>
<td>inserts elements or node (public member function)</td>
</tr>
<tr>
<td>insert_or_assign</td>
<td>inserts an element or as (public member function)</td>
</tr>
<tr>
<td>emplace</td>
<td>constructs element in-pl (public member function)</td>
</tr>
<tr>
<td>emplace_hint</td>
<td>constructs elements in-p (public member function)</td>
</tr>
<tr>
<td>try_emplace</td>
<td>inserts in-place if the key does not exist, does nothing if the key exists (public member function)</td>
</tr>
<tr>
<td>erase</td>
<td>erases elements (public member function)</td>
</tr>
<tr>
<td>swap</td>
<td>swaps the contents (public member function)</td>
</tr>
<tr>
<td>extract</td>
<td>extracts nodes from the container (public member function)</td>
</tr>
<tr>
<td>merge</td>
<td>splices nodes from another container (public member function)</td>
</tr>
</tbody>
</table>

- **count** returns the number of elements matching specific key (public member function)
- **find** finds element with specific key (public member function)
- **contains** checks if the container contains element with specific key (public member function)
- **equal_range** returns range of elements matching a specific key (public member function)
- **lower_bound** returns an iterator to the first element not less than the given key (public member function)
- **upper_bound** returns an iterator to the first element greater than the given key (public member function)
Red-Black Trees in C++

V & std::map<K, V>::operator[]( const K & )

void std::map<K, V>::erase( const K & )
Why Balanced BST?
Summary of Balanced BST

**Pros:**
- Running Time:
  - Improvement Over:

- Great for specific applications:
Summary of Balanced BST

**Cons:**
- Running Time:

- In-memory Requirement:
Trees in the Real World

Q: Can we always fit our data in main memory?

Q: Where else can we keep our data?

We assume constant time memory access, but the constant factor can be limiting in real world settings!
## Memory Hierarchy (Speed of access)

(Measured in 2011 at [https://gist.github.com/hellerbarde/2843375](https://gist.github.com/hellerbarde/2843375))

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (x1 billion)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
<td>0.5 seconds</td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>5 seconds</td>
</tr>
<tr>
<td>L2 cache reference</td>
<td>7 seconds</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>25 seconds</td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100 seconds</td>
</tr>
<tr>
<td>Compress 1K bytes</td>
<td>50 minutes</td>
</tr>
<tr>
<td>Send 2K bytes over 1 Gbps network</td>
<td>5.5 hours</td>
</tr>
<tr>
<td>SSD random read</td>
<td>1.7 days</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>2.9 days</td>
</tr>
<tr>
<td>Read 1 MB sequentially from SSD</td>
<td>11.6 days</td>
</tr>
<tr>
<td>Disk seek</td>
<td>16.5 weeks</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>7.8 months</td>
</tr>
<tr>
<td>Above two together</td>
<td>1 year</td>
</tr>
<tr>
<td>Send packet CA-&gt;Netherlands-&gt;CA</td>
<td>4.8 years</td>
</tr>
</tbody>
</table>

(Values measured in 2011 at [https://gist.github.com/hellerbarde/2843375](https://gist.github.com/hellerbarde/2843375))
AVLs in the Cloud
BTree Motivations

Knowing that we have large seek times for data, we want to:
A BTrees of order \( m \) is an m-way tree:
- All keys within a node are ordered
- All nodes contain no more than \( m-1 \) keys.
BTree in the Real World

Goal: Minimize the number of reads!

Build a tree that uses _______________________ / node
[1 network packet]
[1 disk block]