CS 225
Data Structures

March 5 – AVL Applications
Wade Fagen-Ulmschneider
AVL Runtime Proof

On Friday, we proved an upper-bound on the height of an AVL tree is $2^*\lg(n)$ or $O(\lg(n))$. 
Summary of Balanced BST

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

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

Red-Black Trees
- Max height: $2 \times \lg(n)$
- Constant number of rotations on insert, remove, and find
Why AVL?
Summary of Balanced BST

Pros:
- Running Time:
  - Improvement Over:

- Great for specific applications:
Summary of Balanced BST

Cons:
- Running Time:

- In-memory Requirement:
Red-Black Trees in C++

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

```cpp
std::map<K, V> map;
```
Red-Black Trees in C++

V & std::map<K, V>::operator[]( const K & )
Red-Black Trees in C++

\[ V & \text{ std::map<K, V>::operator[]}(\text{ const K } & ) \]

\[ \text{ std::map<K, V>::erase}(\text{ const K } & ) \]
Red-Black Trees in C++

```cpp
iterator std::map<K, V>::lower_bound( const K & );
iterator std::map<K, V>::upper_bound( const K & );
```
CS 225 -- Course Update

This weekend, the following grades were updated:
• mp1
• mp2*
• mp3*
• lab_inheritance
• lab_quacks
• lab_trees
Iterators

Why do we care?

```cpp
1  DFS dfs(...);
2  for ( ImageTraversal::Iterator it = dfs.begin(); it != dfs.end(); ++it ) {
3      std::cout << (*it) << std::endl;
4  }
```
Iterators

Why do we care?

```cpp
DFS dfs(...);
for ( ImageTraversal::Iterator it = dfs.begin(); it != dfs.end(); ++it ) {
    std::cout << (*it) << std::endl;
}
```

```cpp
DFS dfs(...);
for ( const Point & p : dfs ) {
    std::cout << p << std::endl;
}
```
Iterators

Why do we care?

```cpp
DFS dfs(...);
for ( ImageTraversal::Iterator it = dfs.begin(); it != dfs.end(); ++it ) {
    std::cout << (*it) << std::endl;
}
```

```cpp
DFS dfs(...);
for ( const Point & p : dfs ) {
    std::cout << p << std::endl;
}
```

```cpp
ImageTraversal & traversal = /* ... */;
for ( const Point & p : traversal ) {
    std::cout << p << std::endl;
}
```
Iterators

```cpp
ImageTraversal *traversal = /* ... */;
for ( const Point & p : traversal ) {
    std::cout << p << std::endl;
}
```
### Every Data Structure So Far

<table>
<thead>
<tr>
<th></th>
<th>Unsorted Array</th>
<th>Sorted Array</th>
<th>Unsorted List</th>
<th>Sorted List</th>
<th>Binary Tree</th>
<th>BST</th>
<th>AVL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Find</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Insert</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Remove</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Traverse</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Range-based Searches

Q: Consider points in 1D: $\mathbf{p} = \{p_1, p_2, ..., p_n\}$.
   ...what points fall in [11, 42]?

Tree construction:
Range-based Searches

Balanced BSTs are useful structures for range-based and nearest-neighbor searches.

Q: Consider points in 1D: \( p = \{p_1, p_2, ..., p_n\} \).
   ...what points fall in \([11, 42]\)?

Ex:

\[
\begin{align*}
&3 & & 6 & & 11 & & 33 & & 41 & & 44 & & 55
\end{align*}
\]
Q: Consider points in 1D: \( p = \{p_1, p_2, ..., p_n\} \).

...what points fall in \([11, 42]\)?
Range-based Searches

Q: Consider points in 1D: \( p = \{p_1, p_2, \ldots, p_n\} \).

...what points fall in \([11, 42]\)?

Tree construction:
Range-based Searches
Range-based Searches
Q: Consider points in 1D: \( p = \{p_1, p_2, ..., p_n\} \).  
...what points fall in \([11, 42]\)?
Range-based Searches
Running Time
Q: Consider points in 1D: $p = \{p_1, p_2, ..., p_n\}$.

...what points fall in $[11, 42]$?
Range-based Searches

Consider points in 2D: \( p = \{p_1, p_2, \ldots, p_n\} \).

Q: What points are in the rectangle: 
\[
[ (x_1, y_1), (x_2, y_2) ]
\]

Q: What is the nearest point to \((x_1, y_1)\)?
Range-based Searches

Consider points in 2D: \( p = \{ p_1, p_2, \ldots, p_n \} \).

Tree construction:
Range-based Searches

Diagram showing a set of points labeled $p_1$ to $p_7$ and a tree structure with nodes connected by arrows.
kD-Trees
kD-Trees