#21: AVL Applications
March 5, 2018 · *Wade Fagen-Ulmschneider*

AVL – Proof of Runtime

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

AVL Trees	Red-Black Trees
Balanced BST Max height: 1.44 * lg(n) *Q: Why is our proof 2*lg(n)?* Rotations: - find: - insert: - remove:	Balanced BST *Functionally equivalent to AVL trees; all key operations runs in O(h) time.* Max height: 2 * lg(n) Rotations: - Constant number of rotations for each operation

*In CS 225, we learned **AVL trees** because they're intuitive and I'm certain we could have derived them ourselves given enough time. A red-black tree is simply another form of a balanced BST that is also commonly used.*

Summary of Balanced BSTs:

(Includes both AVL and Red-Black Trees)

Advantages	Disadvantages

Using a Red-Black Tree in C++

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

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

The map implements a dictionary ADT. Primary means of access is through the overloaded `operator[]`:

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

This function can be used for both insert and find!

Removing an element:

```
std::map<K, V>::erase( const K & );
```

Range-based searching:

```
iterator std::map<K, V>::lower_bound( const K & );
iterator std::map<K, V>::upper_bound( const K & );
```

Iterators

Three weeks ago, you saw that you can use an iterator to loop through data:

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

You will use iterators extensively in MP4, creating them in Part 1 and then utilizing them in Part 2. Given the iterator, you can use the for-each syntax available to you in C++:

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

The exact code you might use will have a generic `ImageTraversal`:

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

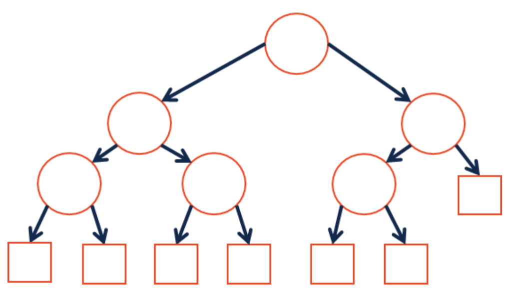
Running Time of Every Data Structure So Far:

	Unsorted Array	Sorted Array	Unsorted List	Sorted List
Find				
Insert				
Remove				
Traverse				

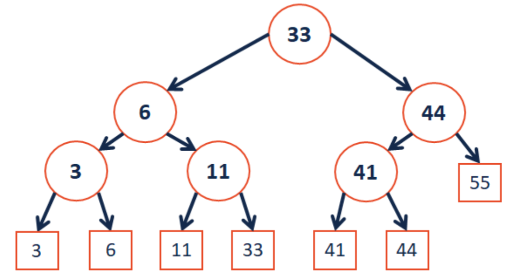
	Binary Tree	BST	AVL
Find			
Insert			
Remove			
Traverse			

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

Tree Construction:

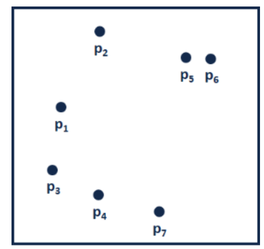


Range-based Searches:



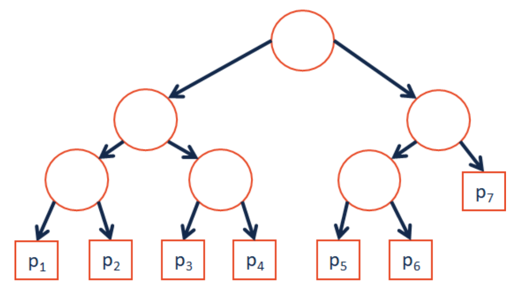
Running Time:

Extending to k-dimensions:
Consider points in 2D: $\mathbf{p = \{p_1, p_2, \ldots, p_n\}}$:



...what points are inside a range (rectangle)?
...what is the nearest point to a query point **q**?

Tree Construction:



CS 225 – Things To Be Doing:

1. Programming Exam B starts next Tuesday (March 13^{th})
2. MP4 extra credit due tonight (Monday, March 5^{th})
3. lab_avl released this week; details regarding labs on Wednesday
4. Daily POTDs are ongoing!