CS 225

Data Structures

March 9 – AVL Applications

G Carl Evans

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))**:

N(h) := 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

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

# of nodes (n) ≥ N(h) > $2^{h/2}$

n > $2^{h/2}$

lg(n) > h/2

2 × lg(n) > h

h < 2 × lg(n) ***, for h ≥ 1***

Proved: The maximum number of nodes in an AVL tree of height h is less than 2 × lg(n).

Summary of Balanced BST

AVL Trees

- Max height: 1.44 * 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).

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:

Red-Black Trees in C++

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

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

Red-Black Trees in C++

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

Red-Black Trees in C++

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

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

Red-Black Trees in C++

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

CS 225 -- Course Update

Your grades can now be viewed on moodle (https://learn.illinois.edu/)

We will discuss the grades for the course as a whole (ex: average, etc) in lecture on Wednesday.

Iterators

Why do we care?

```
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?

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

```
1  DFS dfs(...);
2  for ( const Point & p : dfs ) {
3    std::cout << p << std::endl;
4  }
```

Iterators

Why do we care?

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

```
1  DFS dfs(...);
2  for ( const Point & p : dfs ) {
3    std::cout << p << std::endl;
4  }
```

```
1  ImageTraversal & traversal = /* ... */;
2  for ( const Point & p : traversal ) {
3    std::cout << p << std::endl;
4  }
```

Every Data Structure So Far

	Unsorted Array	Sorted Array	Unsorted List	Sorted List	Binary Tree	BST	AVL
Find							
Insert							
Remove							
Traverse							

Range-based Searches

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

…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: $\mathbf{p = \{p_1, p_2, \ldots, p_n\}}$.

…what points fall in [11, 42]?

Ex:

Range-based Searches

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

…what points fall in [11, 42]?

Ex:

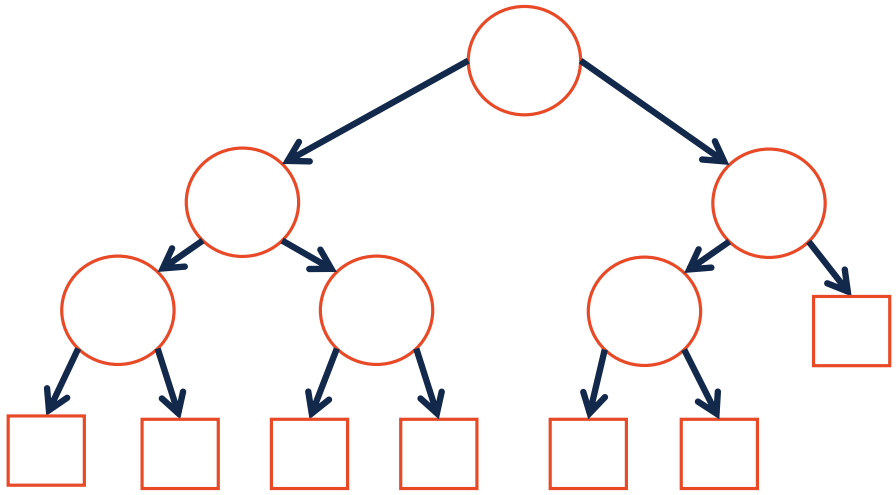
Range-based Searches

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

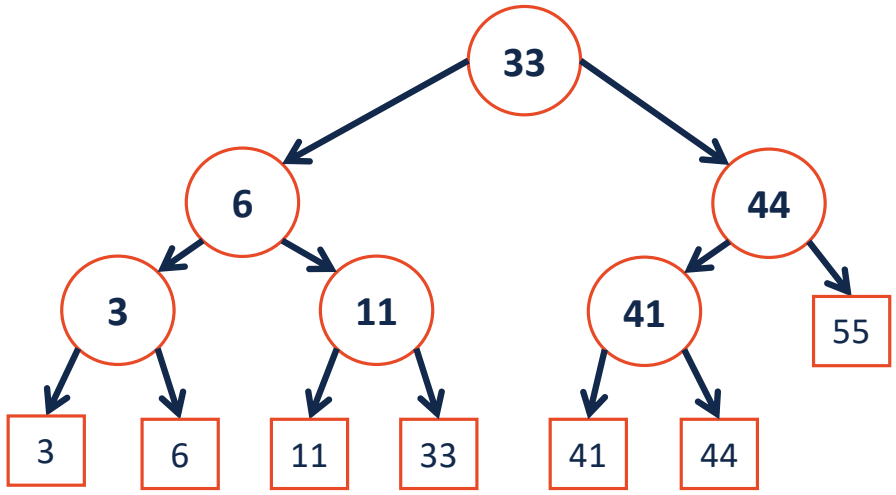
…what points fall in [11, 42]?

Tree construction:

Range-based Searches



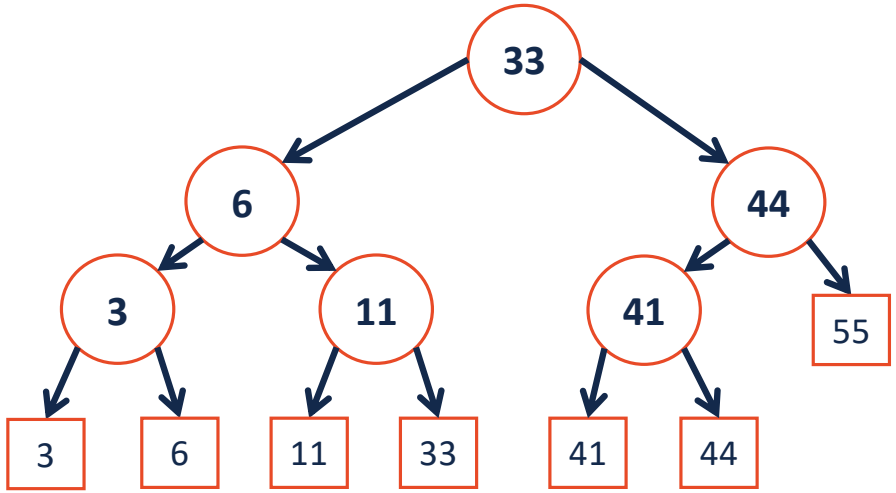
Range-based Searches



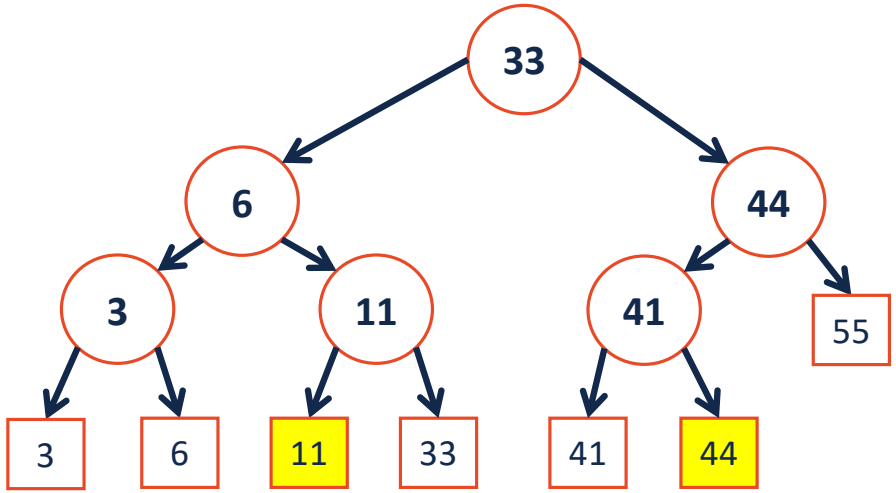
Range-based Searches

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

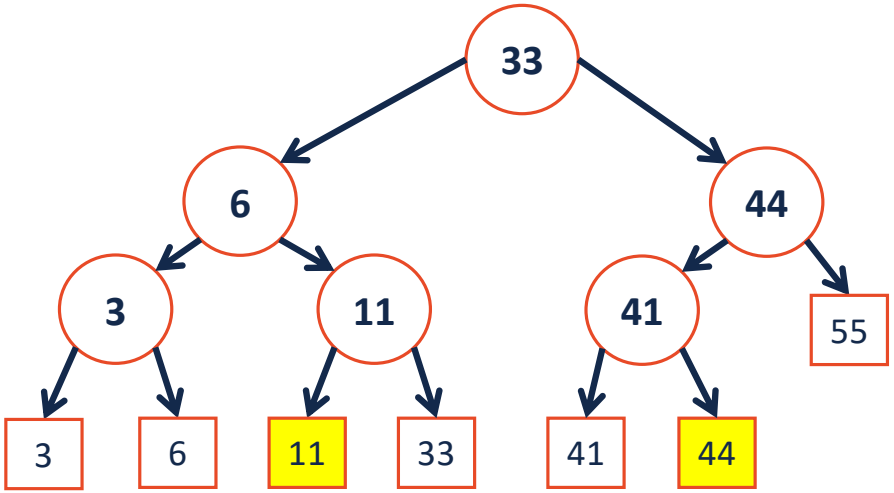
…what points fall in [11, 42]?



Range-based Searches



Running Time



Range-based Searches

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

…what points fall in [11, 42]?

Ex:

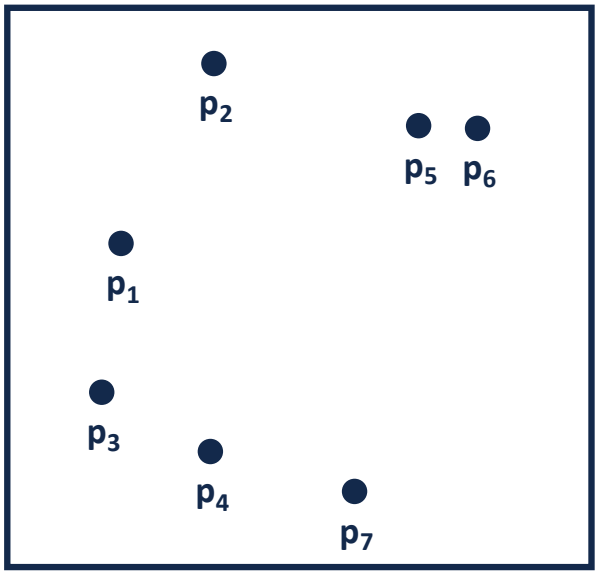
Range-based Searches

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

Q: What points are in the rectangle:

$\mathbf{[\ (x_1, y_1),\ (x_2, y_2)\ ]?}$

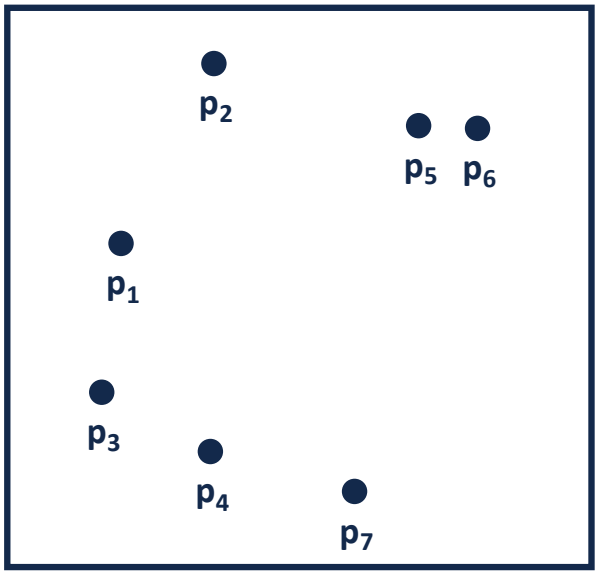
Q: What is the nearest point to $\mathbf{(x_1, y_1)}$?



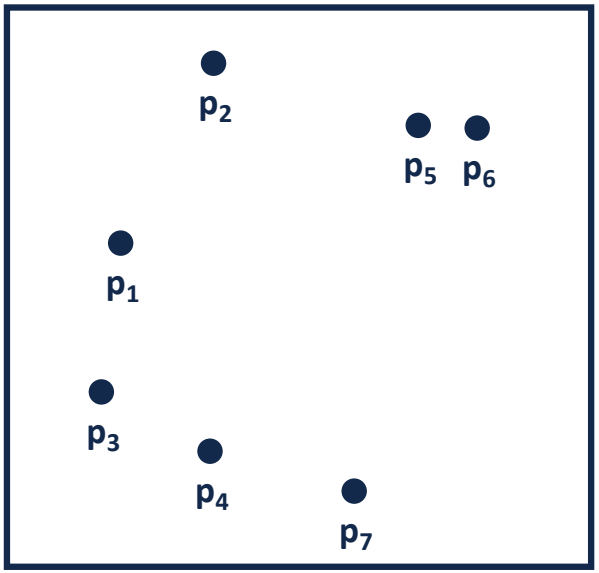
Range-based Searches

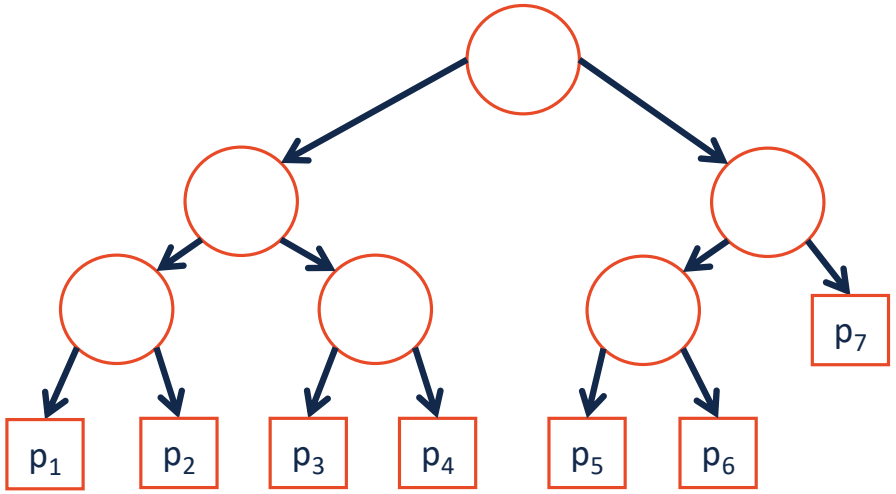
Consider points in 2D: $\mathbf{p = \{p_1, p_2, \ldots, p_n\}}$.

Tree construction:

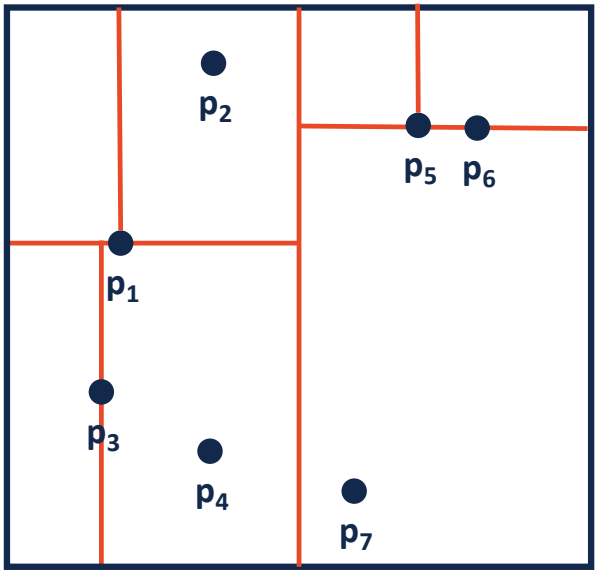


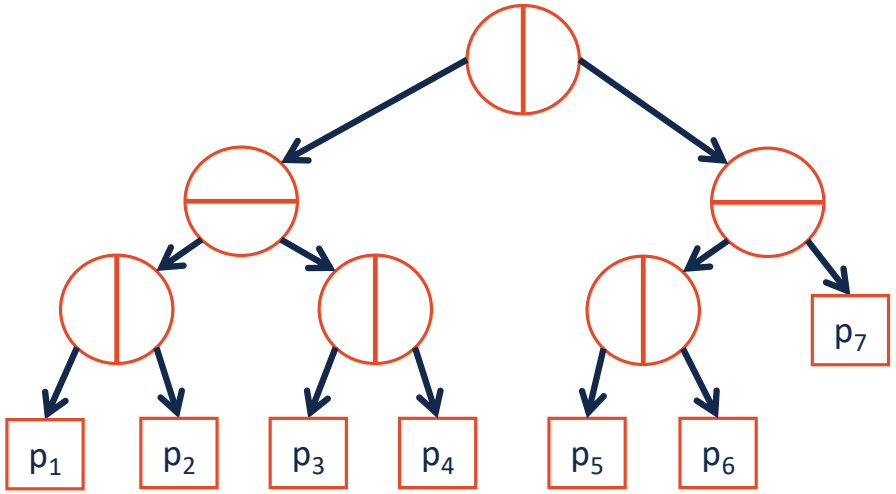
Range-based Searches





kD-Trees





kD-Trees