Binary Search Trees

CS 62 - Spring 2016
Michael Bannister

Binary Search Tree

A **binary search tree** is a binary tree that is either empty or every value in its left subtree is less than (or equal to) the value of the root and every value inits right subtree is greater than the value of the root.

Used as a more efficient sorted list!

BST Interface*

```
public class BinarySearchTree<E extends Comparable<E>> {
  // Public methods
  public void add(E value) // O(log n)
  public E remove(E value) // O(log n)
  public boolean contains(E value) // O(log n)
  public E get(E value) // O(log n)
  public boolean isEmpty() // O(1)
  public void clear() // O(1)
  public int size() // O(1

  // Important helpers
  protected BinaryTree<E> locate(BinaryTree<E> root, E value)
  protected BinaryTree<E> predecessor(BinaryTree<E> root)
  protected BinaryTree<E> successor(BinaryTree<E> root)
  protected BinaryTree<E> removeTop(BinaryTree<E> topNode)
}
```

Locate Value

- If tree is empty, then not found!
- If **value** is at root, then found!
- If **value** ≤ root, call locate on left subtree.
- If **value** > root, call locate on right subtree

When **value** is not in the tree, locate finds where it "should" be!

Predecessor and Successor

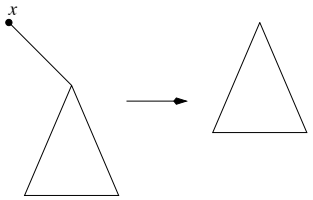
- Predecessor of root
  - In left subtree go right as far as possible
- Successor of root
  - In right subtree go left as far as possible

Add Value

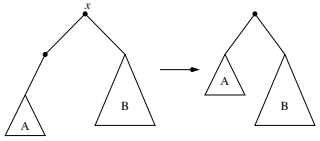
- Use locate to find position of where the **value** should be, and insert node with **value**.
- If **value** is already in the tree, then more work is needed to adjust tree. See code and example on the board.

Remove node

- Remove topmost node.



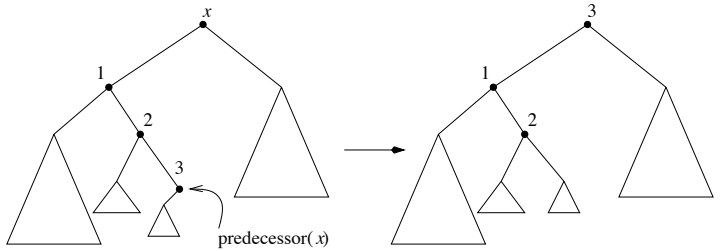
- Easy cases:
  - no left subtree, or no right subtree -- easy, they are new tree



  - left child has no right subtree

General Case

- Left Child has a right subtree:



Remove method

- Locate element to be deleted
- RemoveTop of node rooted at element
- Hook up resulting tree as child of elt's parent.
- O(h), where h is height of tree.
  - O(h) to find,
  - Could be another O(h) to find predecessor
  - Constant to patch back together.

Complexity

- Runtimes depend on the height of the tree.
- To achieve a O(log n) runtime we need to keep the tree "balanced".

Tree Rotations

- Change the structure of the BST while preserving the ordering properties.