CS 310: BST Find and Insert

Chris Kauffman

Week 11-2
Logistics

Reading
Weiss Ch. 19 BSTs

HW 3
- Due Sunday
- Last chance to dance: any questions?

Today
- BST insert()
- BST remove()
Last Time

Tree Traversals

- What are tree traversals and how many did we discuss?
- Time and Space complexity of recursive traversals?
- Any other traversals you can think of

Tree Iterators

- Want a post-order tree iterator with next() and remove() methods
- How can one be implemented

Binary Search Tree (BST)

- What makes a Binary Tree a BST?
- How does implement find(t) in a BST?
Quick Note on Generics

From Weiss

public class BinarySearchTree<T extends Comparable<? super T>>

The type

T extends Comparable<? super T>

means *descends from something Comparable vs.* T extends Comparable<T>

comparable to self only
Examples of Insert

Play with MyBST.java in JGrasp and look at the pretty pictures after multiple insert(x) calls.
Insertion: Similar to $\text{find}(x)$

- May need to change a left or right pointers, redefine root
- No duplication, define a TreeSet, exception on duplicate insert

**Define Recursive Insert**

class BinarySearchTree<T> {
    Node<T> root=null; int size=0;
    public void insert( T x ){
        root = insert( x, root );
    }
    protected Node<T> insert( T x, Node<T> t ){
        // DEFINE ME
    }
}

**Define Iterative Insert if You’re Brave**

public void insert( T x ){
    // DEFINE ME
}
Recursive `insert(x,t)`

From `weiss/nonstandard/BinarySearchTree.java`

class BinarySearchTree<T> {
    Node<T> root;
    public void insert( T x ){
        root = insert( x, root );
    }
    Node<T> insert( T x, Node<T> t )
    {
        if( t == null )
            t = new Node<T>( x );
        else if( x.compareTo( t.data ) < 0 )
            t.left = insert( x, t.left );
        else if( x.compareTo( t.data ) > 0 )
            t.right = insert( x, t.right );
        else
            throw new DuplicateItemException( x.toString( ) );
        return t;
    }
}
class BinarySearchTree<T> {
    Node<T> root;
    public void insert(T x) {
        if (this.root == null) {
            this.root = new Node<T>(x);
            return;
        }
        Node<T> t = this.root;
        while (true) {
            int diff = x.compareTo(t.data);
            if (diff == 0) {
                throw new DuplicateItemException(x.toString());
            }
            if (diff < 0) {
                if (t.left == null) {
                    t.left = new Node<T>(x);
                    return;
                } else {
                    t = t.left;
                }
            } else {
                if (t.right == null) {
                    t.right = new Node<T>(x);
                    return;
                } else {
                    t = t.right;
                }
            }
        }
    }
}
Binary Search Tree remove(x)

// Public method, eliminate x if present in tree
public void remove(T x);

// Recursive helper method
private Node<T> remove(T x, Node<T> t);

Get rid of a node with data x in a binary tree

- More involved than find/insert
- Preserve Tree Structure
- Recursion greatly eases implementation
Consider Mario Tree

- What cases are of interest for `remove()`?
- What is the appropriate new tree for each case?