

Introduction to Computer Science II

CS112-2008S-34

More Binary Search Trees

David Galles

Department of Computer Science
University of San Francisco

34-0: Binary Trees

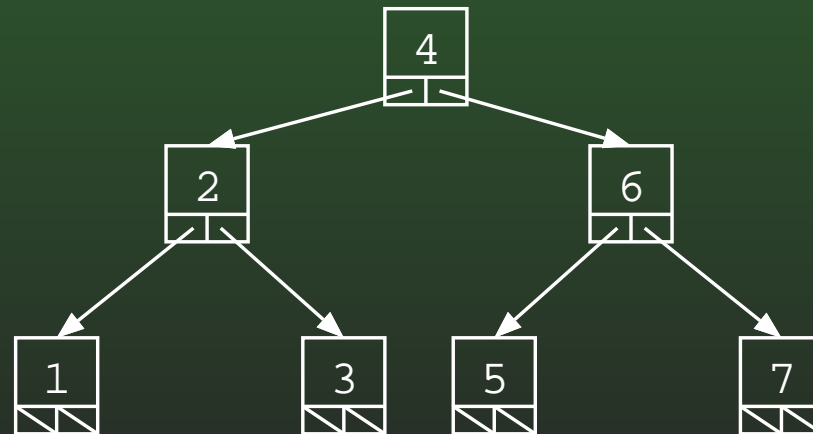
Binary Trees are Recursive Data Structures

- Base Case: Empty Tree
- Recursive Case: Node, consisting of:
 - Left Child (Tree)
 - Right Child (Tree)
 - Data

34-1: Binary Search Trees

- Binary Trees
- For each node n , (value stored at node n) \geq (value stored in left subtree)
- For each node n , (value stored at node n) $<$ (value stored in right subtree)

34-2: Example Binary Search Trees



34-3: Binary Search Trees

```
class BSTNode
{
    private int data;
    private BSTNode left;
    private BSTNode right;

    public BSTNode(int data, BSTNode left, BSTNode right)
    {
        this.left = left;
        this.right = right;
        this.data = data;
    }
    public BSTNode left()
    {
        return left;
    }
    public BSTNode right()
    {
        return right;
    }
    public int data()
    {
        return data;
    }
}
```

34-4: Binary Search Trees

```
class BSTNode
{
    ...
    public BSTNode left()           public void setLeft(BSTNode newLeft)
    {                               {
        return left;               left = newLeft;
    }                               }
    public BSTNode right()         public void setRight(BSTNode newRight)
    {                               {
        return left;               right = newRight;
    }                               }
    public int data()              public void setData(int newdata)
    {                               {
        return data;               data = newdata;
    }                               }
}
}
```

34-5: Finding an Element in a BST

To find an element e in a Binary Search Tree T :

- If T is empty, then e is not in T
- If the root of T contains e , then e is in T
- If $e <$ the element stored in the root of T :
 - Look for e in the left subtree of T

Otherwise

- Look for e in the right subtree of T

34-6: Finding an Element in a BST

```
boolean find(BSTNode tree, int elem) {
    if (tree == null)
        return false;
    if (elem == tree.data())
        return true;
    if (elem < tree.data())
        return find(tree.left(), elem);
    else
        return find(tree.right(), elem);
}
```

34-7: Printing out a BST

To print out all element in a BST:

- Print all elements in the left subtree, in order
- Print out the element at the root of the tree
- Print all elements in the right subtree, in order

34-8: Printing out a BST

To print out all element in a BST:

- Print all elements in the left subtree, in order
- Print out the element at the root of the tree
- Print all elements in the right subtree, in order
 - Each subproblem is a smaller version of the original problem – we can assume that a recursive call will work!

34-9: Printing out a BST

- What is the base case for printing out a Binary Search Tree – what is an easy tree to print out?

34-10: Printing out a BST

- What is the base case for printing out a Binary Search Tree – what is an easy tree to print out?
- An empty tree is extremely easy to print out – do nothing!
- Code for printing a BST ...

34-11: Printing out a BST

```
void print(Node tree) {  
    if (tree != null) {  
        print(tree.left());  
        System.out.println(tree.element());  
        print(tree.right());  
    }  
}
```

34-12: Inserting e into BST T

- What is the base case – an easy tree to insert an element into?

34-13: Inserting e into BST T

- What is the base case – an easy tree to insert an element into?
 - An empty tree
 - Create a new tree, containing the element e

34-14: Inserting e into BST T

- Recursive Case: How do we make the problem smaller?

34-15: Inserting e into BST T

- Recursive Case: How do we make the problem smaller?
 - The left and right subtrees are smaller versions of the same problem.
 - How do we use these smaller versions of the problem?

34-16: Inserting e into BST T

- Recursive Case: How do we make the problem smaller?
 - The left and right subtrees are smaller versions of the same problem
 - Insert the element into the left subtree if $e \leq$ value stored at the root, and insert the element into the right subtree if $e >$ value stored at the root

34-17: Inserting e into BST T

- Base case – T is empty:
 - Create a new tree, containing the element e
- Recursive Case:
 - If e is less than the element at the root of T , insert e into left subtree
 - If e is greater than the element at the root of T , insert e into the right subtree

34-18: Implementing Insert I

- Write a function that inserts an element into a BST
- For now, we will assume that the tree we are inserting into is not empty
 - Later, we can special-case the empty tree
- Like in linked lists, we can't keep going until we reach a null tree
 - We a pointer to the node *before* the null tree

34-19: Implementing Insert I

```
void insert (BSTNode tree, int elem)
{
    if (elem <= tree.data() &&
        tree.left() == null)
    {
        currentNode.setLeft(new BSTNode(elem, null, null));
    }
    else if (data > tree.data() &&
            tree.right() == null)
    {
        currentNode.setRight = (new BSTNode(elem, null, null));
    }
    else if (elem <= tree.data)
    {
        insert(tree.left(), elem);
    }
    else
    {
        insert(tree.right(), elem)
    }
}
```

34-20: Implementing Insert I

- The previous code was uglier than it needed to be
 - Need to special-case insertion into an empty tree
 - Stopping right before reaching an empty tree (instead of stopping after having reached the empty tree) makes for extra cases

34-21: Tree Manipulation in Java

- Tree manipulation functions return trees
- Insert method takes as input the old tree and the element to insert, and returns the new tree, with the element inserted
 - Old value (pre-insertion) of tree will be destroyed
- To insert an element e into a tree T :
 - `T = insert(T, e);`

34-22: Inserting e into BST T

```
Node insert(BSTNode tree, Comparable elem) {
    if (tree == null)
    {
        return new BSTNode(elem, null, null);
    }
    if (elem <= tree.data())
    {
        tree.setLeft(insert(tree.left(), elem));
        return tree;
    } else {
        tree.setRight(insert(tree.right(), elem));
        return tree;
    }
}
```

34-23: Deleting From a BST

- Removing a leaf:
 - Remove element immediately
- Removing a node with one child:
 - Just like removing from a linked list
 - Make parent point to child
- Removing a node with two children:
 - Replace node with largest element in left subtree, or the smallest element in the right subtree

34-24: Implementation Details

- Create an ordered list class
 - Insert, remove, find, print
- Implement as a BST
- Hide implementation details from user

How can we do it?

34-25: Implementation Details

```
class OrderedList
{
    private BSTNode root;

    public void insert(int data)
    {
        root = insert(root, data);
    }

    private BSTNode insert(BSTNode tree, int data)
    {
        if (tree == null)
        {
            return new BSTNode(elem, null, null);
        }
        if (elem <= tree.data())
        {
            tree.setLeft(insert(tree.left(), elem));
            return tree;
        } else {
            tree.setRight(insert(tree.right(), elem));
        }
    }
}
```