L17: Binary Trees & Tree Recursion

Alex Steiger

CompSci 201: Spring 2024

3/18/2024

Announcements, Coming up

- Wednesday 3/20
  - Midterm 2, linked list through 3/4 + Binary Search from 3/6
  - Practice exams available on Sakai resources
- Next Monday 3/25
  - Project P4: Autocomplete due
- Next Wednesday 3/27
  - APT 7 (tree recursion problems) due

Midsemester Survey

- Thanks!
- Results: ~60% completion rate, wanted >70%
- Exam 2 Extra Credit:
  - +1 pt to everyone
    - Feedback is insightful and greatly appreciated
  - +1 pt to everyone who submitted

Today's Agenda

1. Binary Trees
    1. Definitions
    2. Binary *Search* Trees

2. Tree Recursion problems
    1. TreeCount
    2. HeightLabel
    3. Diameter

Binary Trees

Comparing TreeSet/Map with HashSet/Map

TreeSet/Map

- O(log(N)) add, contains, put, get ***are <u>not</u> amortized***.
- Stored in sorted order
  - Natural ordering by default; can provide Comparator
- Can get range of values in sorted order efficiently

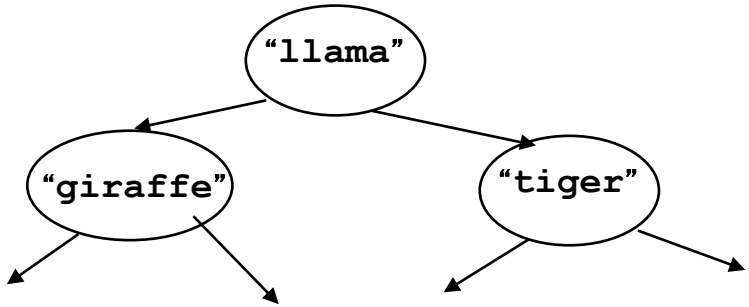
HashSet/Map

- O(1) add, contains, put, get, are ***amortized***.
- Unordered data structures
- Cannot get range efficiently, stored unordered

TreeNode to store Strings

```
public class TreeNode {
    TreeNode left;
    TreeNode right;
    String info;
    TreeNode(String s, TreeNode llink, TreeNode rlink){
        info = s;
        left = llink;
        right = rlink;
    }
}
```

Like LinkedList but each node has 2 references/pointers instead of 1



APT TreeNode to store ints

APT TreeNode will only hold integer. Would need to create another class to hold Strings? Another for…?

```
public class TreeNode {
    int info;
    TreeNode left;
    TreeNode right;
    TreeNode(int x){
        info = x;
    }
    TreeNode(int x, TreeNode lNode, TreeNode rNode){
        info = x;
        left = lNode;
        right = rNode;
    }
}
```

FAQ: Making a tree with nodes?

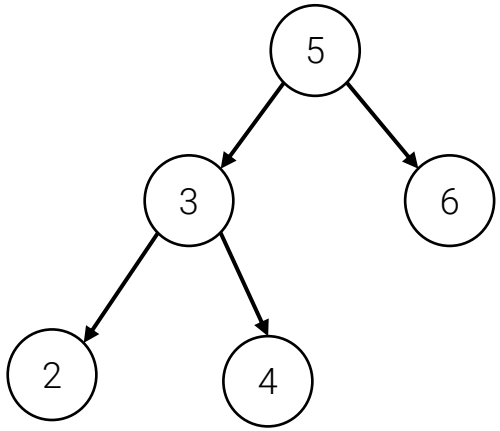
```
public class TreeNode {
    int info;
    TreeNode left;
    TreeNode right;
    TreeNode(int x){
        info = x;
    }
    TreeNode(int x, TreeNode lNode, TreeNode rNode){
        info = x;
        left = lNode;
        right = rNode;
    }
```

Just call the TreeNode constructor for each new node and connect them.

More terse version

```
TreeNode myTree = new TreeNode( x: 5,
        new TreeNode( x: 3,
                new TreeNode( x: 2),
                new TreeNode( x: 4)),
        new TreeNode( x: 6));
```

```
TreeNode root = new TreeNode( x: 5);
root.left = new TreeNode( x: 3);
root.right = new TreeNode( x: 6);
root.left.left = new TreeNode( x: 2);
root.left.right = new TreeNode( x: 4);
```



Aside: Generic TreeNode?

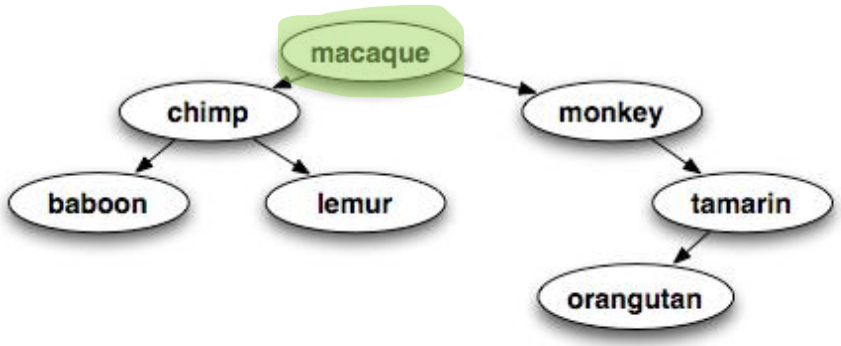
```
1   public class TreeNode<T> {
2       T info;
3       TreeNode<T> left;
4       TreeNode<T> right;
5       TreeNode(T x){
6           info = x;
7       }
8       TreeNode(T x, TreeNode<T> lNode, TreeNode<T> rNode){
9           info = x;
10          left = lNode;
11          right = rNode;
12      }
```

```
14      public static void main(String[] args) {
15          TreeNode<String> sTree = new TreeNode<>("hi");
16          TreeNode<Integer> iTree = new TreeNode<>(201);
```

Generics allow us to write one kind of Node (or List, or Set, …) that can hold different types.

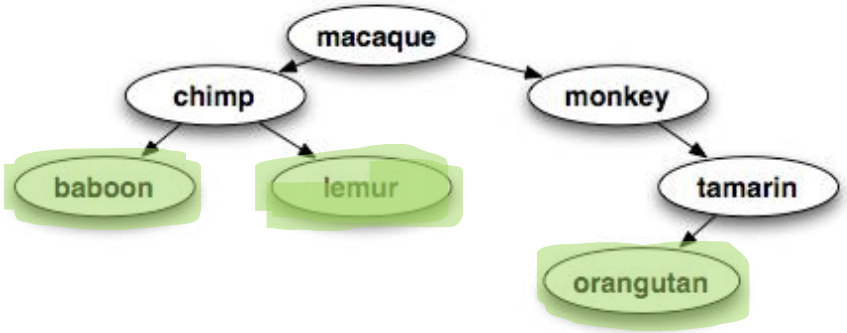
Tree terminology

- *Root*: "top node", has no parent, node you pass for the whole tree/subtree.
  - **Example**: “macaque”
- *Leaf*: “bottom” nodes, have no children / both `null`
  - Example: “orangutan”
- *Path*: sequence of parent-child nodes
  - Example: "macaque", "chimp", "lemur”
- *Subtree*: nodes at and beneath
  - ”chimp”, “baboon”, “lemur”



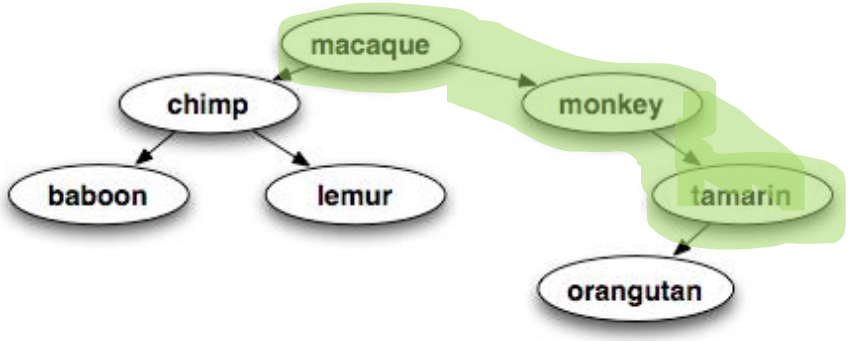
Tree terminology

- *Root*: "top node", has no parent, node you pass for the whole tree/subtree.
  - **Example**: “macaque”
- *Leaf*: “bottom” nodes, have no children / both `null`
  - Example: “orangutan”
- *Path*: sequence of parent-child nodes
  - Example: "macaque", "chimp", "lemur”
- *Subtree*: nodes at and beneath
  - ”chimp”, “baboon”, “lemur”



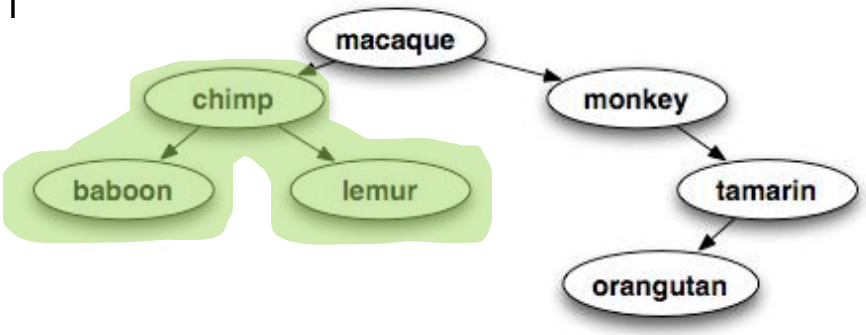
Tree terminology

- *Root*: "top node", has no parent, node you pass for the whole tree/subtree.
  - **Example**: “macaque”
- *Leaf*: “bottom” nodes, have no children / both `null`
  - Example: “orangutan”
- *Path*: sequence of parent-child nodes
  - Example: "macaque", "chimp", "lemur”
- *Subtree*: nodes at and beneath
  - ”chimp”, “baboon”, “lemur”



Tree terminology

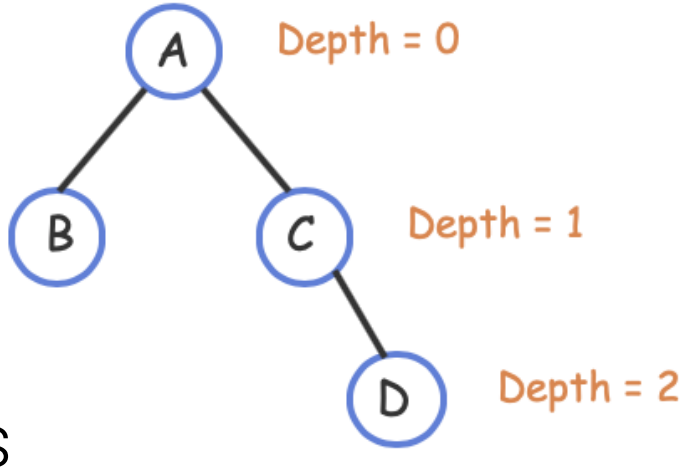
- *Root*: "top node", has no parent, node you pass for the whole tree/subtree.
  - **Example**: “macaque”
- *Leaf*: “bottom” nodes, have no children / both `null`
  - Example: “orangutan”
- *Path*: sequence of parent-child nodes
  - Example: "macaque", "chimp", "lemur”
- *Subtree*: nodes at and beneath
  - ”chimp”, “baboon”, “lemur”



More tree terminology

The **depth** of a node is the number of edges from the root to the node.

The **height** of a tree is the maximum depth of any node.



- (Sometimes defined as maximum number of nodes on any root-to-leaf path
- = 1 + max depth.)

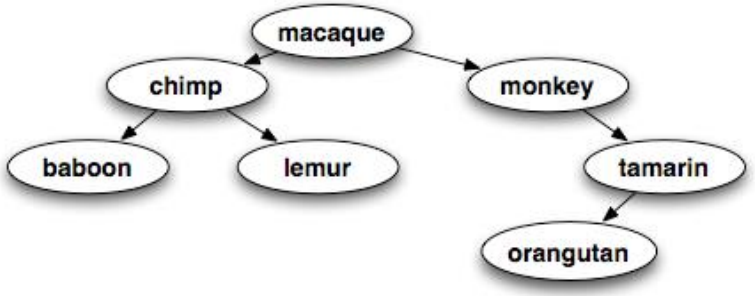
In-Order Traversal

- How to “loop over” nodes in a tree?
  - One option: In-order traversal and visit/print/process
  - Search tree values printed “in order”
    - Left subtree, then current node, then right subtree

baboon, chimp, lemur	macaque	monkey, orangutan, tamarin

baboon, chimp, lemur, macaque, monkey, orangutan, tamarin

```
49      public void inOrder(TreeNode root) {
50          if (root != null) {
51              inOrder(root.left);
52              System.out.println(root.info);
53              inOrder(root.right);
54          }
55      }
```



Helper method to return List of nodes' info

```
101     public ArrayList<String> visit(TreeNode root) {
102         ArrayList<String> list = new ArrayList<>();
103         doInOrder(root,list);
104         return list;
105     }
106
107     private void doInOrder(TreeNode root, ArrayList<String> list) {
108         if (root!= null) {
109             doInOrder(root.left,list);
110             list.add(root.info);
111             doInOrder(root.right,list);
112         }
113     }
```

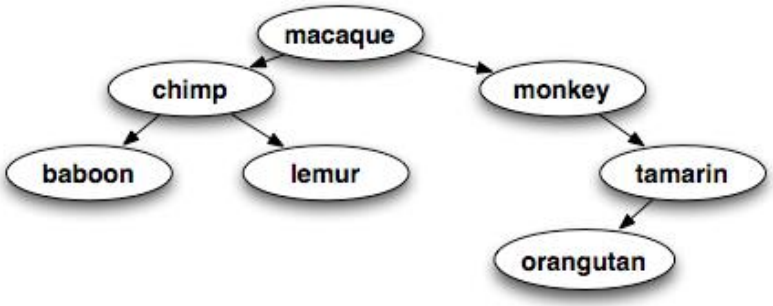
- In order traversal → Store in a list?
  - Similar to prev. slide, but add nodes to a list instead of print
- Create empty list, call helper with list, then return it
- Values in returned list are in traversal order

Three ways to recursively traverse a tree

- Difference is in where the ***non-recursive*** part is

inorder	preorder	psotorder

`void inOrder(TreeNode t) {`
`  if (t != null) {`
`  inOrder(t.left);`
`  System.out.println(t.info);`
`  inOrder(t.right);`
`  }`
`}`
	`void preOrder(TreeNode t) {`
	`  if (t != null) {`
	`  System.out.println(t.info);`
	`  preOrder(t.left);`
	`  preOrder(t.right);`
	`  }`
	`}`
		`void postOrder(TreeNode t) {`
		`  if (t != null) {`
		`  postOrder(t.left);`
		`  postOrder(t.right);`
		`  System.out.println(t.info);`
		`  }`
		`}`

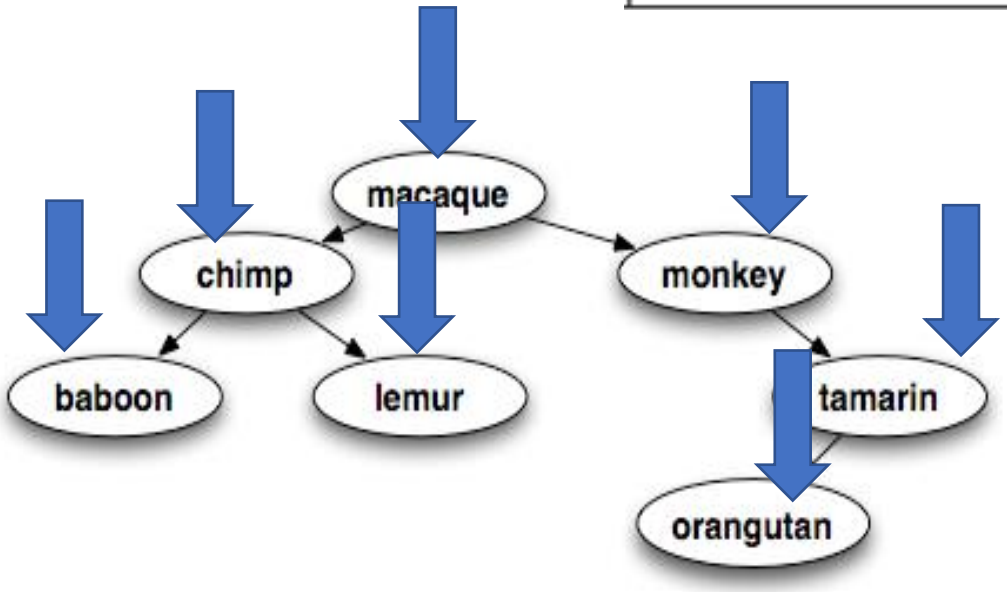


Preorder Traversal

- macaque
- chimp
- baboon
- lemur
- monkey
- tamarin
- orangutan

preorder

```
void preOrder(TreeNode t) {
  if (t != null) {
    System.out.println(t.info);
    preOrder(t.left);
    preOrder(t.right);
  }
}
```



Binary Search Tree Invariant

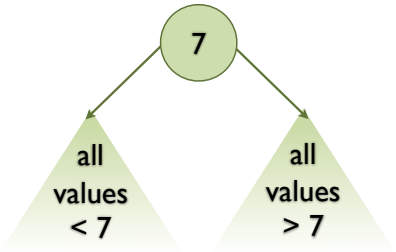
A binary tree is a binary ***search*** tree if *for every node*:

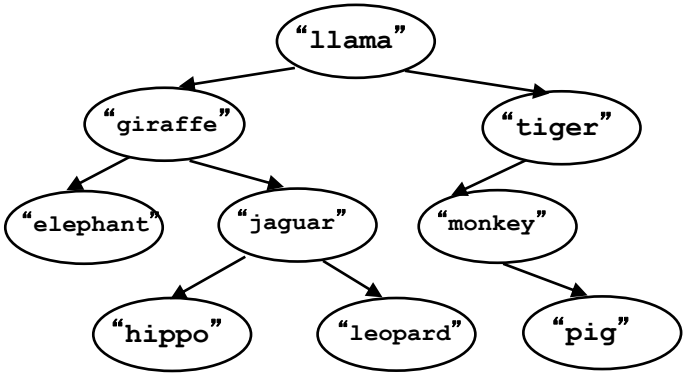
- Left subtree values are all ***less than*** the node's value

AND

- Right subtree values are all ***greater than*** the node's value

According to some ordering (natural ordering if Comparable or defined by Comparator)

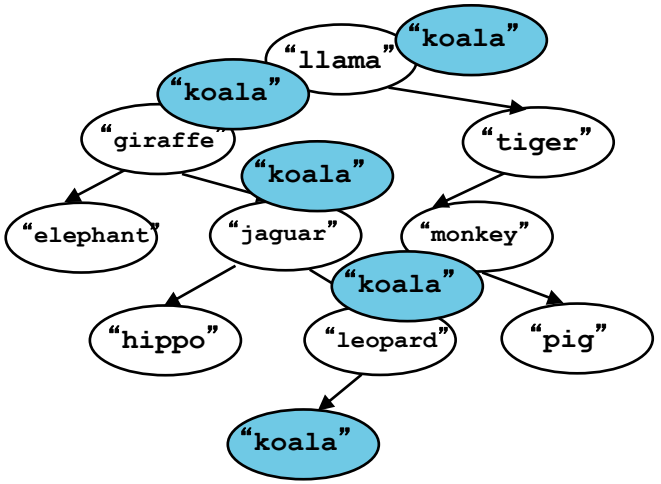




Enables efficient search, similar to binary search!

Recursive Search in Binary Search Tree

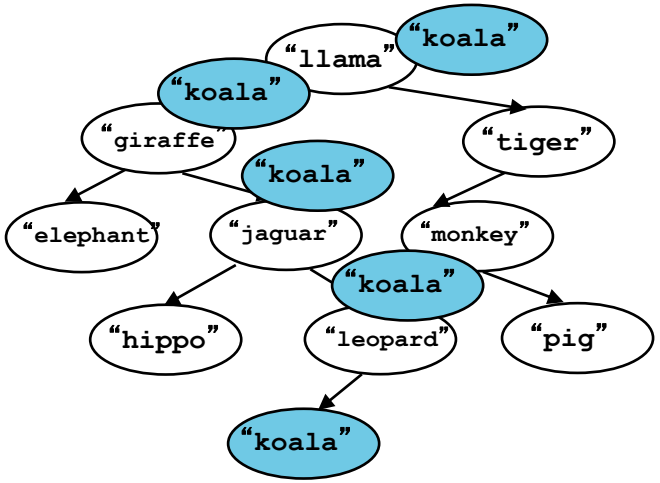
- Code for search
  - Insertion is very similar
  - **`target.compareTo(…)`**




```
186     public boolean contains(TreeNode tree, String target) {
187         if (tree == null) return false;
188         int result = target.compareTo(tree.info);
189         if (result == 0) return true;
190         if (result < 0) return contains(tree.left,target);
191         return contains(tree.right, target);
192     }
```

Iterative search in binary search tree

```
48  // assumes node is a search tree, else may return false negatives
49  public static boolean contains(TreeNode<String> node, String target) {
50      while (node != null) {
51          int comp = node.info.compareTo(target);
52          if (comp == 0) {
53              return true;
54          }
55          else if(comp > 0) {
56              node = node.left;
57          }
58          else {
59              node = node.right;
60          }
61      }
62      return false;
63  }
```



Again, insertion is very similar

L17-WOTO1-SearchTree-Sp24

Hi, Alexander. When you submit this form, the owner will see your name and email address.

* Required

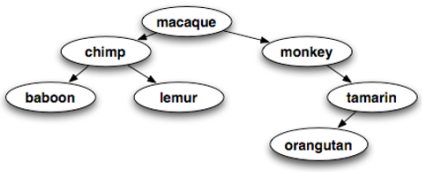
1

NetID *

Enter your answer

2

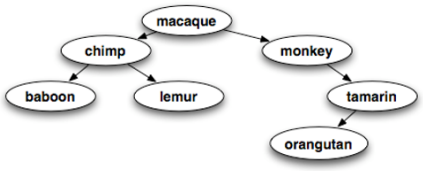
If we define the root to have depth 0 and the height of a tree to be the maximum depth of any node, then the height of the tree shown is... *



- ◯ 0
- ◯ 1
- ◯ 2
- ⊗ 3
- ◯ 4

3

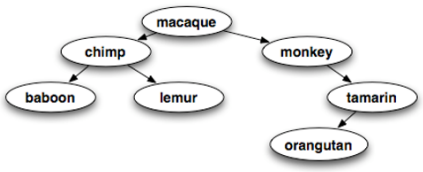
The leaves of the tree shown are... *



- [ ] baboon, chimp, lemur, monkey, orangutan, tamarin
- [ ] baboon, lemur, monkey, orangutan, tamarin
- [x] baboon, lemur, orangutan
- [ ] orangutan

4

The subtree rooted at monkey has how many nodes? *



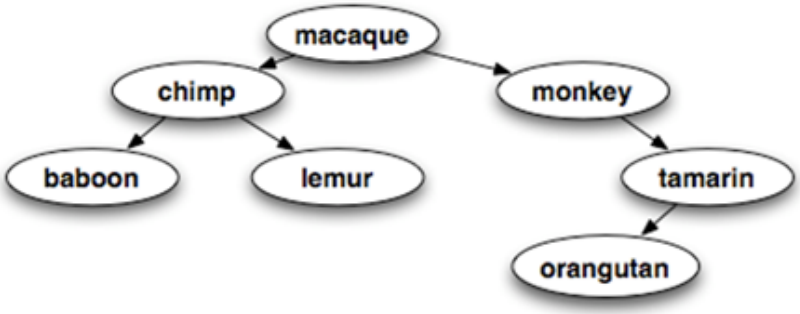
- [ ] 2
- [x] 3
- [ ] 4
- [ ] 7

5

Printing the values of this tree using a **post-order** traversal of this tree would print... *

psotorder

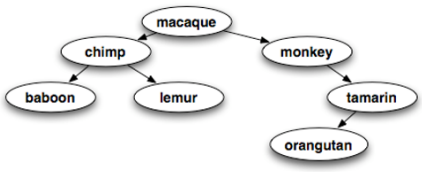
```
void postOrder(TreeNode t) {
  if (t != null) {
    postOrder(t.left);
    postOrder(t.right);
    System.out.println(t.info);
  }
}
```



- ◯ baboon, chimp, lemur, macaque, monkey, orangutan, tamarin
- ☒ baboon, lemur, chimp, orangutan, tamarin, monkey, macaque
- ◯ macaque, chimp, baboon, lemur, monkey, tamarin, orangutan

6

If "capuchin" is added and the tree is still a search tree, where is it added? *



- ◯ left child of lemur
- ☒ right child of baboon
- ◯ right child of lemur
- ◯ left child of baboon

Tree Recursion and Problem-Solving

Tree Recursion tips / common mistakes

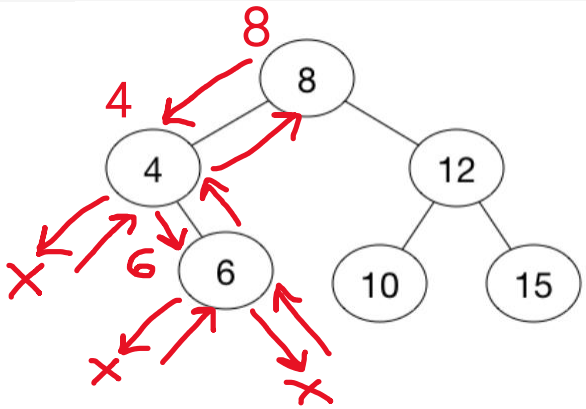
1. Draw it out! Trace your code on small examples.
2. Return type of the method. Do you need a helper method?
3. Base case first, otherwise infinite recursion / null pointer exception.
4. If you make a recursive call, (usually) make sure to use what it returns.

TreeCount APT and pre-order string representation

Problem Statement

Write a method that returns the number of nodes of a binary tree. The TreeNode class will be accessible when your method is tested.

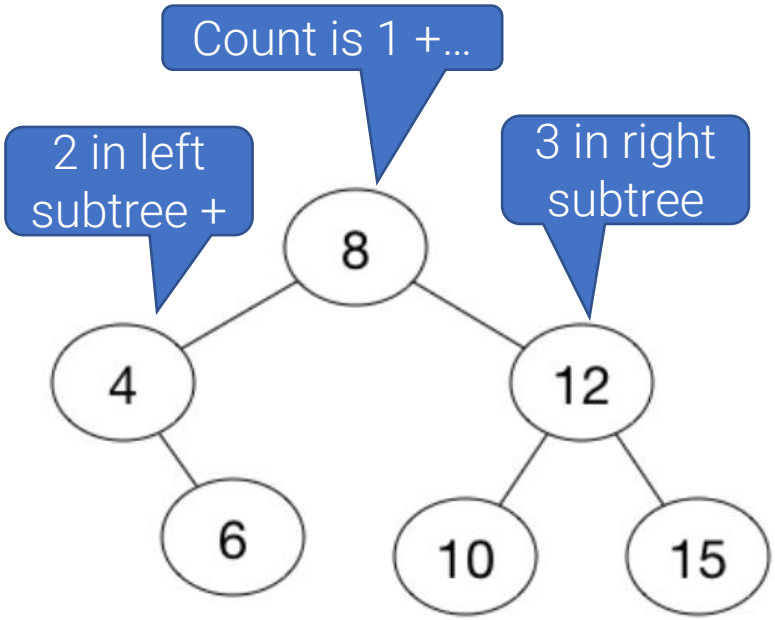
```
public class TreeCount {
    public int count(TreeNode tree) {
        // replace with working code
        return 0;
    }
}
```



is characterized by the pre-order string **`8, 4, x, 6, x, x, 12, 10, x, x, 15, x, x`**

Live Coding TreeCount

Solving TreeCount in Picture & Code

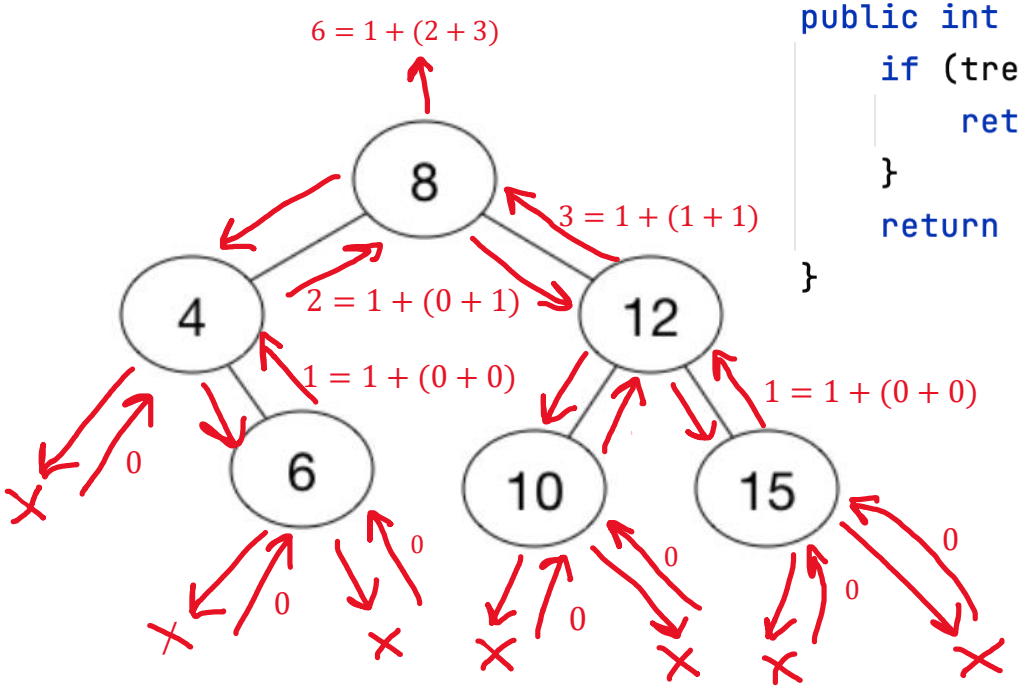


Base case: 0 nodes in an empty tree / null

Recursive case:
- 1 (count current node)
- + count of left subtree
- + count of right subtree

```
public int count(TreeNode tree) {
    if (tree == null) {
        return 0;
    }
    return 1 + count(tree.left) + count(tree.right);
}
```

Messy Details of TreeCount Solution




```
public int count(TreeNode tree) {
    if (tree == null) {
        return 0;
    }
    return 1 + count(tree.left) + count(tree.right);
}
```

Analyzing Recursive Runtime

Develop a recurrence relation of the form

$$T(N) = a \cdot T\big(g(N)\big) + f(N)$$

Total runtime

Recursive call(s)

Non-recursive runtime

Where:

- $T(N)$ - runtime of method with input size *N*
- a is the number of recursive calls
- $g(N)$ - how much input size decreases on each recursive call
- $f(N)$ - runtime of non-recursive code on input size N

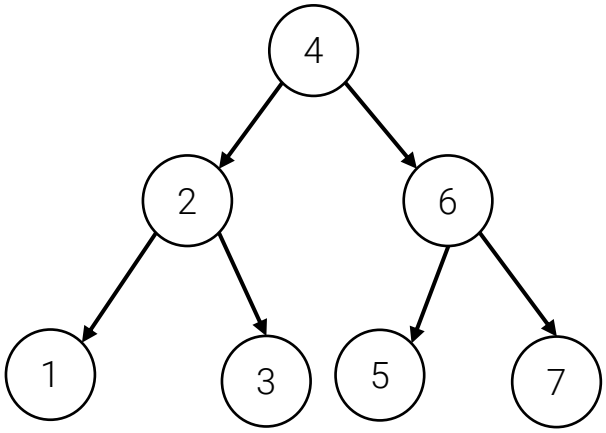
Table of Recurrences

Recurrence	Algorithm	Solution
`T(n) = T(n/2) + O(1)`	binary search	`O(log n)`
`T(n) = T(n-1) + O(1)`	sequential search	`O(n)`
`T(n) = 2T(n/2) + O(1)`	tree traversal	`O(n)`
`T(n) = T(n/2) + O(n)`	qsort partition ,find k^{th}	`O(n)`
`T(n) = 2T(n/2) + O(n)`	mergesort, quicksort	`O(n log n)`
`T(n) = T(n-1) + O(n)`	selection or bubble sort	$O(n^2)$

We expect you to be able to derive a recurrence relation from an algorithm, but not necessarily to solve. We will provide a table of solutions like this for exams.

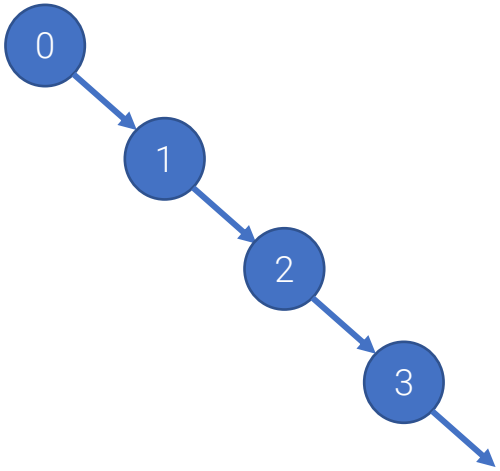
Balance and Trees

Balanced



For each node, left and right subtrees have roughly equal number of nodes.

Unbalanced



One subtree has many more nodes than the other.

Recurrence relation and runtime for traversing a ***balanced*** tree

- *T(n)* time for **`count(tree)`** with *n* nodes (balanced)

```
public int count(TreeNode tree) {
    if (tree == null) {
        return 0;
    }
    return 1 + count(tree.left) + count(tree.right);
}
```

n/2 nodes in this subtree

n/2 nodes in this subtree

- T(n) = 2T(n/2) + O(1)
- = O(n)

Recurrence relation and runtime for traversing *unbalanced* tree

- *T(n)* time for **`count(tree)`** with *n* nodes (unbalanced)

```
public int count(TreeNode tree) {
    if (tree == null) {
        return 0;
    }
    return 1 + count(tree.left) + count(tree.right);
}
```

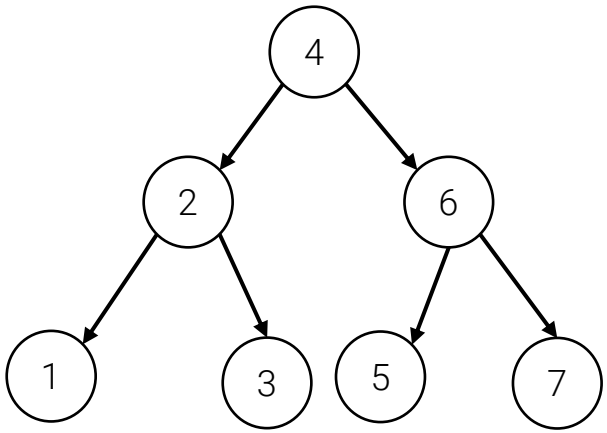
1 node in this subtree

n-1 nodes in this subtree

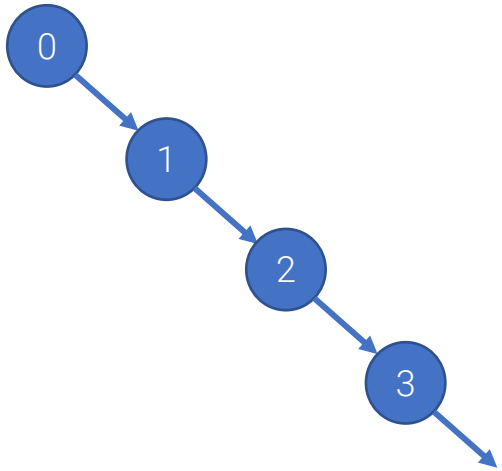
- T(n) = T(1) + T(n-1) + O(1)
- = O(1) + T(n-1) + O(1)
- = O(n)

Balance Binary Search Tree Runtime (add, contains)

Balanced



Unbalanced



$T(n) = T(n/2) + O(1)$
$= O(\log(n))$

We will return to this problem later!

$T(n) = T(n-1) + O(1)$
$= O(n)$