L17: Binary Trees & Tree Recursion

Alex Steiger

CompSci 201: Spring 2024

3/18/2024

1

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

3

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

4

Today's Agenda

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

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

5

Binary Trees

6

Comparing TreeSet/Map with HashSet/Map

TreeSet/Map

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

HashSet/Map

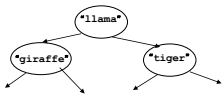
- O(1) add, contains, put, get, are ***amortized***.
- Unordered data structures
- Cannot get range efficiently, stored unordered

7

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



8

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;
    }
}
```


9

FAQ: Making a tree with nodes?

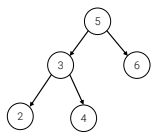
```
public class TreeNode {
    int info;
    TreeNode left;
    TreeNode right;
    TreeNode(int x){
        info = x;
    }
    TreeNode(int x, TreeNode lNode, TreeNode rNode){
        info = x;
        left = lNode;
        right = rNode;
    }
```

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

```
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);
```

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));
```



10

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      }
```

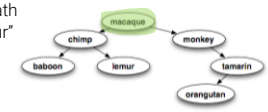
Generics allow us to write one kind of Node (or List, or Set, …) that can hold different types.

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


11

Tree terminology

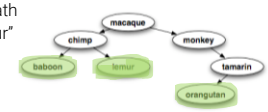
- *Root*: "top node", has no parent, node you pass for the whole tree/subtree.
  - **Example**: "macaque"
- *Leaf*: "bottom" nodes, have no children / both `null`
  - Example: "orangutan"
- *Path*: sequence of parent-child nodes
  - Example: "macaque", "chimp", "lemur"
- *Subtree*: nodes at and beneath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



12

Tree terminology

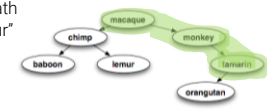
- *Root*: "top node", has no parent, node you pass for the whole tree/subtree.
  - **Example**: "macaque"
- *Leaf*: "bottom" nodes, have no children / both `null`
  - Example: "orangutan"
- *Path*: sequence of parent-child nodes
  - Example: "macaque", "chimp", "lemur"
- *Subtree*: nodes at and beneath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



13

Tree terminology

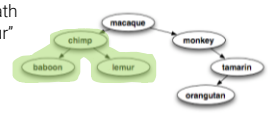
- *Root*: "top node", has no parent, node you pass for the whole tree/subtree.
  - **Example**: "macaque"
- *Leaf*: "bottom" nodes, have no children / both `null`
  - Example: "orangutan"
- *Path*: sequence of parent-child nodes
  - Example: "macaque", "chimp", "lemur"
- *Subtree*: nodes at and beneath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



14

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"



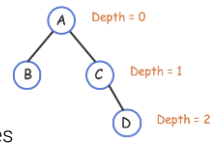
15

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.)



16

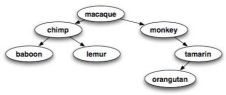
In-Order Traversal

- 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  }
```



17

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

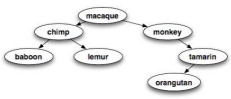
18

Three ways to recursively traverse a tree

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

inorder	preorder	postorder

`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);`
		`  }`
		`}`

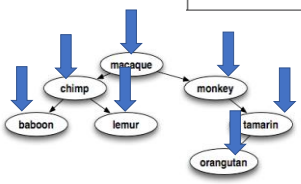


19

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);
  }
}
```



20

Binary Search Tree Invariant

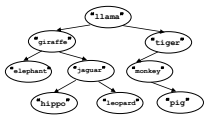
A binary tree is a binary ***search*** tree if *for every node*:

- 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)

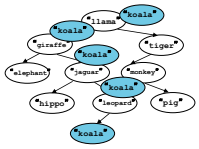


Enables efficient search, similar to binary search!

21

Recursive Search in Binary Search Tree

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

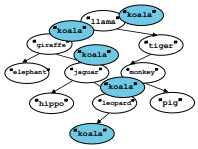



```
186     public boolean contains(TreeNode tree, String target) {
187         if (tree == null) return false;
188         int result = target.compareTo(tree.info);
189         if (result == 0) return true;
190         if (result < 0) return contains(tree.left,target);
191         return contains(tree.right,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


22

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

23

Tree Recursion and Problem-Solving

25

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.

26

L17-WOTO1-SearchTree-Sp24

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

* Required

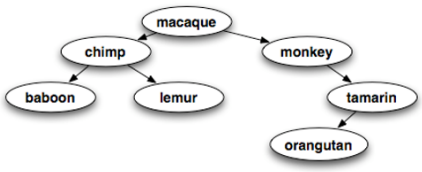
1

NetID *

Enter your answer

2

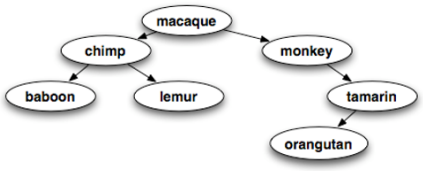
If we define the root to have depth 0 and the height of a tree to be the maximum depth of any node, then the height of the tree shown is... *



- ◯ 0
- ◯ 1
- ◯ 2
- ⊗ 3
- ◯ 4

3

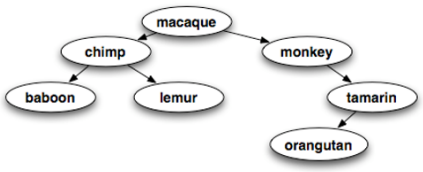
The leaves of the tree shown are... *



- [ ] 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? *



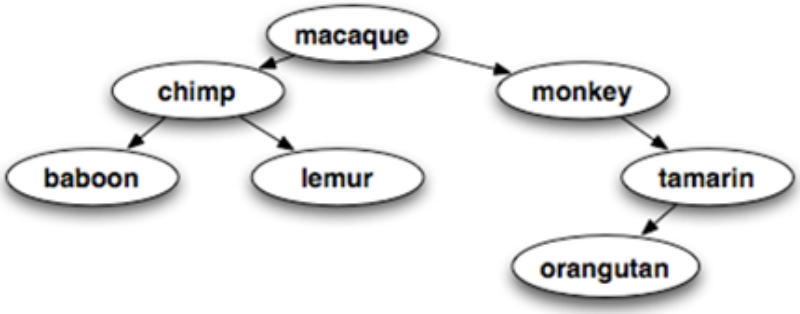
- [ ] 2
- [x] 3
- [ ] 4
- [ ] 7

5

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

psotorder

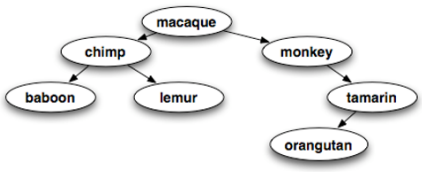
```
void postOrder(TreeNode t) {
  if (t != null) {
    postOrder(t.left);
    postOrder(t.right);
    System.out.println(t.info);
  }
}
```



- ◯ 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

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

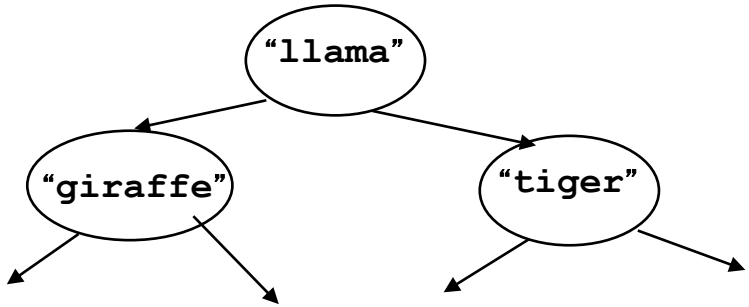
HashSet/Map

- 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?

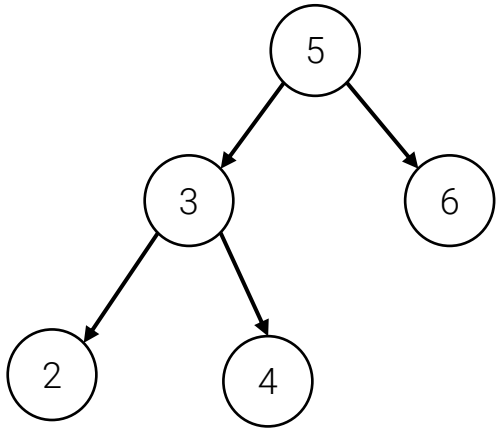
```
public class TreeNode {
    int info;
    TreeNode left;
    TreeNode right;
    TreeNode(int x){
        info = x;
    }
    TreeNode(int x, TreeNode lNode, TreeNode rNode){
        info = x;
        left = lNode;
        right = rNode;
    }
```

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?

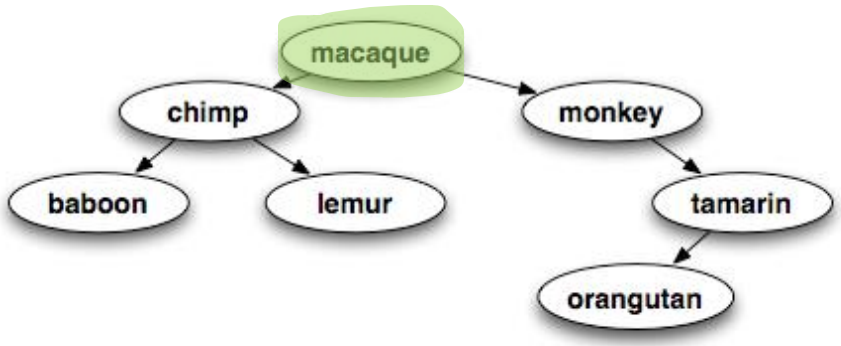
```
1   public class TreeNode<T> {
2       T info;
3       TreeNode<T> left;
4       TreeNode<T> right;
5       TreeNode(T x){
6           info = x;
7       }
8       TreeNode(T x, TreeNode<T> lNode, TreeNode<T> rNode){
9           info = x;
10          left = lNode;
11          right = rNode;
12   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

```
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.

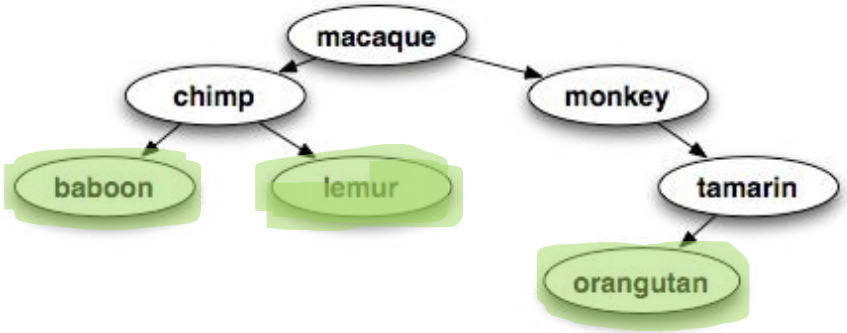
Tree terminology

- *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”



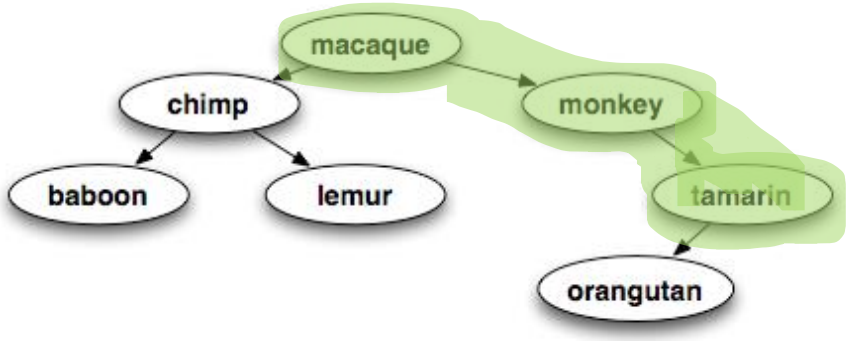
Tree terminology

- *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”



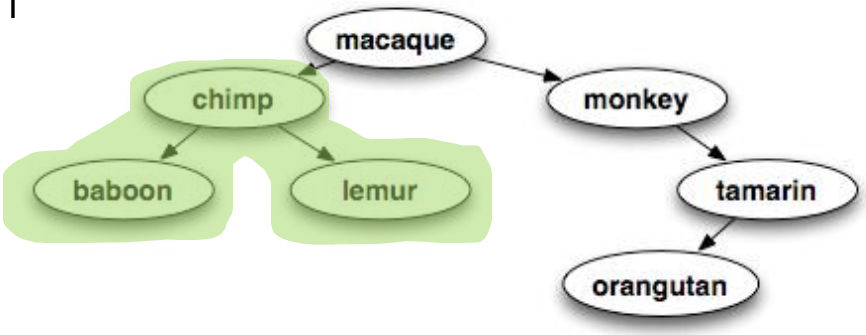
Tree terminology

- *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

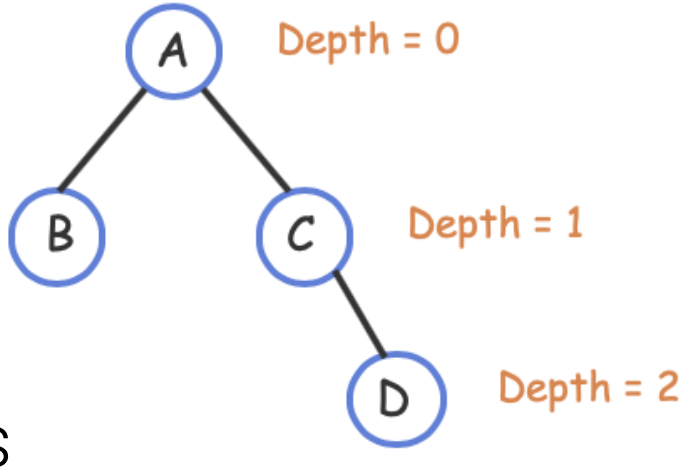
- *Root*: "top node", has no parent, node you pass for the whole tree/subtree.
  - **Example**: “macaque”
- *Leaf*: “bottom” nodes, have no children / both `null`
  - Example: “orangutan”
- *Path*: sequence of parent-child nodes
  - Example: "macaque", "chimp", "lemur”
- *Subtree*: nodes at and beneath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

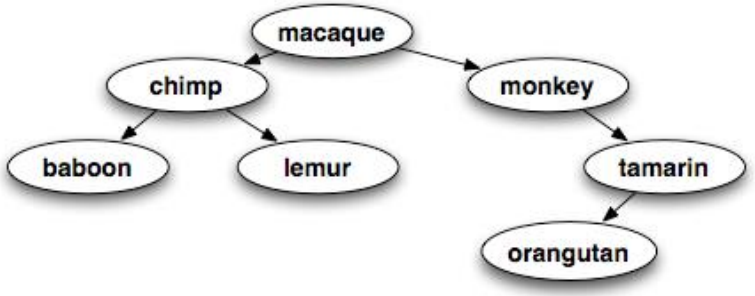
In-Order Traversal

- 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     }
```

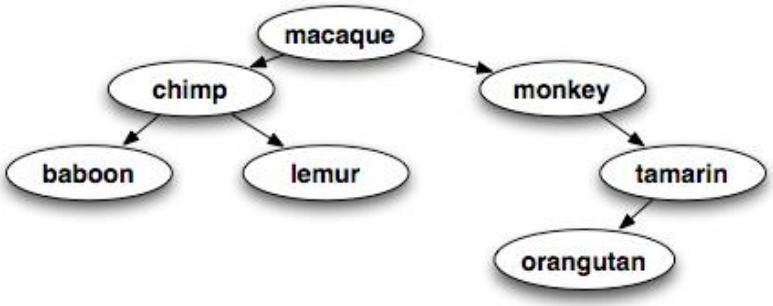
- In order traversal → Store in a list?
  - Similar to prev. slide, but add nodes to a list instead of print
- Create empty list, call helper with list, then return it
- Values in returned list are in traversal order

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);`
		`  }`
		`}`

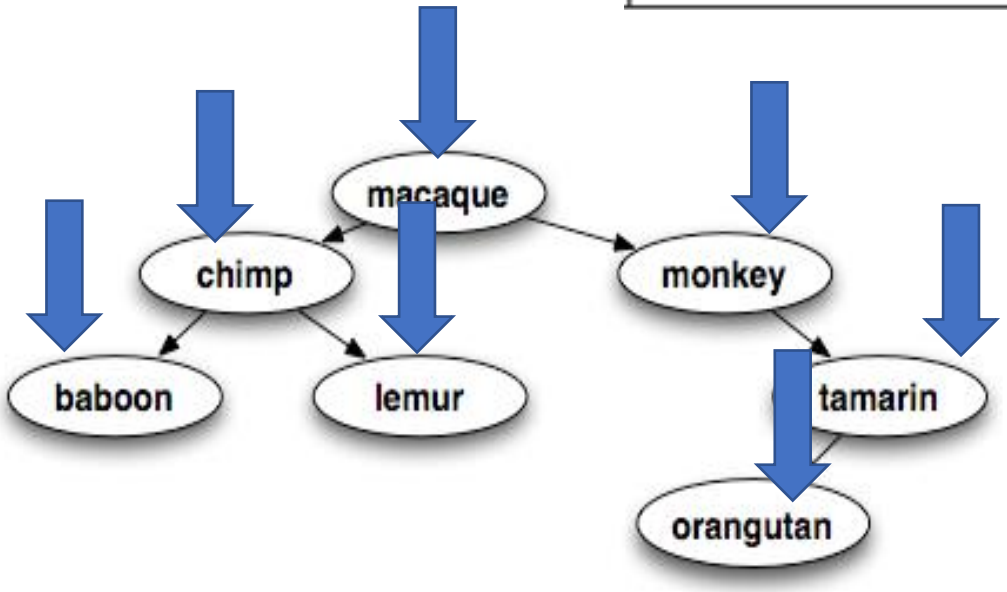


Preorder Traversal

- 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

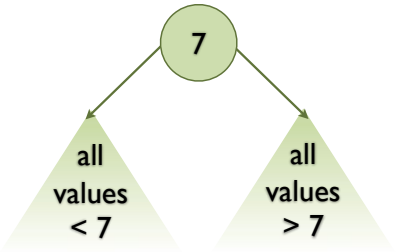
A binary tree is a binary ***search*** tree if *for every node*:

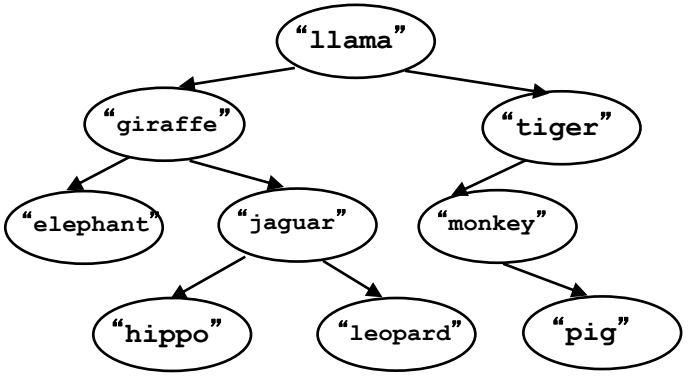
- Left subtree values are all ***less than*** the node's value

AND

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

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

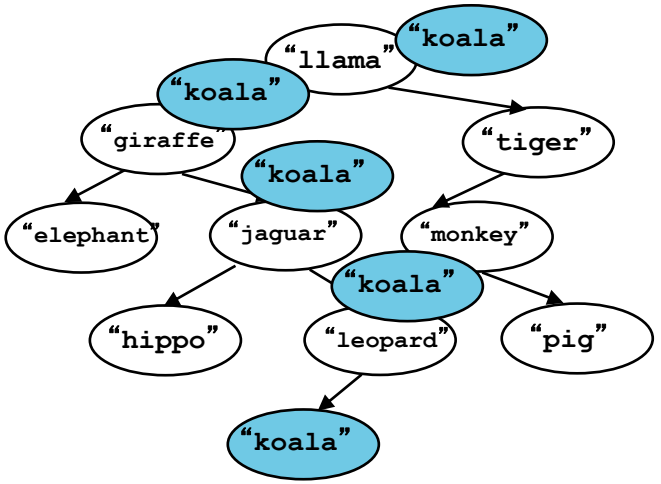




Enables efficient search, similar to binary search!

Recursive Search in Binary Search Tree

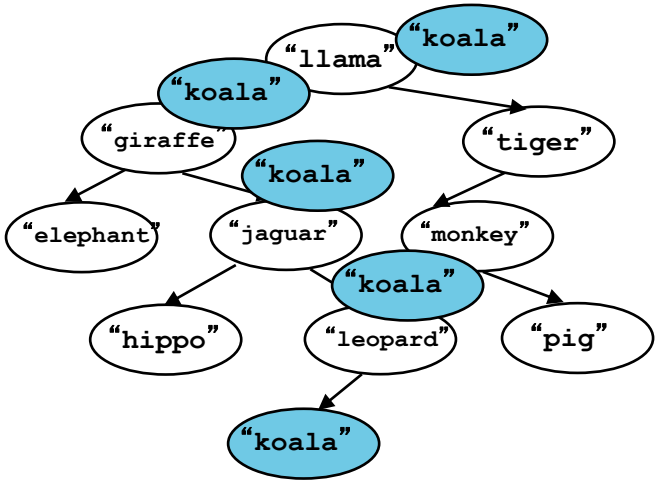
- Code for search
  - Insertion is very similar
  - **`target.compareTo(…)`**




```
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