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

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
TreeNode to store Strings

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

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...

```java
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?

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

```
TreeNode root = new TreeNode(5);
root.left = new TreeNode(3);
root.right = new TreeNode(0);
root.left.left = new TreeNode(2);
root.left.right = new TreeNode(4);
```

More terse version:

```
TreeNode myTree = new TreeNode(5, new TreeNode(3, new TreeNode(2), new TreeNode(4)), new TreeNode(0));
```
Aside: Generic TreeNode?

```java
public class TreeNode<T> {
    T info;
    TreeNode<T> left;
    TreeNode<T> right;
    <Node, TreeNode<T> <Mode>[
        info = x;
        left = xNode;
        right = xNode;
    ]
```

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"

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

```
public void inorderTraversal(TreeNode root) {
    if (root != null) {
        inorderTraversal(root.left);
        System.out.print(root.data + " ");
        inorderTraversal(root.right);
    }
}
```

Helper method to return List of nodes' info

```
public ArrayList<String> visit(TreeNode root) {
    ArrayList<String> list = new ArrayList<>();
    inorderTraversal(root);
    return list;
}
```

- 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
Preorder Traversal

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

---

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(...)`

```java
public boolean containsTreeNode(String target) {
    if (tree == null) return false;
    int result = target.compareTo(tree.info);
    if (result == 0) return true;
    if (result < 0) return containsTreeNode(tree.left, target);
    return containsTreeNode(tree.right, target);
}
```

---
Iterative search in binary search tree

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

Again, insertion is very similar

---

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

* Required

NetID *

Enter your answer

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...

*
3

The leaves of the tree shown are... *

- baboon, chimp, lemur, monkey, orangutan, tamarin
- baboon, lemur, monkey, orangutan, tamarin
- baboon, lemur, orangutan
- orangutan

4

The subtree rooted at monkey has how many nodes? *

- 2
- 3
- 4
- 7

5

Printing the values of this tree using a **post-order** traversal of this tree would print... *
baboon, chimp, lemur, macaque, monkey, orangutan, tamarin

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

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 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
TreeNode to store Strings

```java
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...?

```java
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?

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

```java
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;
    }
}
```

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?

```java
public class TreeNode<T> {
    T info;
    TreeNode<T> left;
    TreeNode<T> right;
    TreeNode(T x) {
        info = x;
    }
    TreeNode(T x, TreeNode<T> lNode, TreeNode<T> rNode) {
        info = x;
        left = lNode;
        right = rNode;
    }

    public static void main(String[] args) {
        TreeNode<String> sTree = new TreeNode<>("hi");
        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)
- \( = 1 + \text{max depth.} \)
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

public void inOrder(TreeNode root) {
    if (root != null) {
        inOrder(root.left);
        System.out.println(root.info);
        inOrder(root.right);
    }
}
Helper method to return List of nodes’ info

public ArrayList<String> visit(TreeNode root) {
    ArrayList<String> list = new ArrayList<>();
    doInOrder(root, list);
    return list;
}

private void doInOrder(TreeNode root, ArrayList<String> list) {
    if (root != null) {
        doInOrder(root.left, list);
        list.add(root.info);
        doInOrder(root.right, list);
    }
}

• 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 \textit{non-recursive} part is
Preorder Traversal

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

preorder

```java
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(…)`

```java
public boolean contains(TreeNode tree, String target) {
    if (tree == null) return false;
    int result = target.compareTo(tree.info);
    if (result == 0) return true;
    if (result < 0) return contains(tree.left, target);
    return contains(tree.right, target);
}
```
Iterative search in binary search tree

```java
// assumes node is a search tree, else may return false negatives
public static boolean contains(TreeNode<String> node, String target) {
    while (node != null) {
        int comp = node.info.compareTo(target);
        if (comp == 0) {
            return true;
        } else if (comp > 0) {
            node = node.left;
        } else {
            node = node.right;
        }
    }
    return false;
}
```

Again, insertion is very similar.