CSE 143: Computer Programming II

Binary Trees

Outline

1 LinkedLists to BinaryTrees
2 Why Do We Care About Binary Trees?
3 Printing Recursively
4 Binary Tree Traversals

LinkedLists to BinaryTrees

Why Do We Care About Binary Trees?

Printing Recursively

Binary Tree Traversals

Back To LinkedLists

Consider the following standard LinkedList:

```
front
```

Back To LinkedLists

Nodes with Multiple next Fields

```
front
```

Introducing Trees

Binary Trees

```
root
```

```
front
```

```
root
```

```
front
```

What if we added more fields?
- Multiple data fields?
- Multiple "next" fields?

```
front
```

```
front
```

```
front
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```

```
root
```
Consider the following LinkedList of a mathematical expression:

( 2 ∗ 4 ) + ( 7 − 3 )

What’s bad about it?
- It doesn’t really help us with the structure
- Looking at it doesn’t really show us what’s going on

What about this structure instead?

Now we can see the order of operations much more clearly!

**Uses of Trees**

- Parsing (Programming Languages, Math, etc.)
- Implementing TreeSet
- Directory File Structure

**More Uses of Trees**

- Recursive Trees (including things like games of Tic-Tac-Toe)
- Compression (this will be your last assignment!)

**Printing a LinkedList (Again)**

To print a LinkedList...
- Print the front of the list
- Print the next of the list (recursively)

```java
public void print() {
    print(this.front);
}

public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.next);
    }
}
```

**Printing a Tree Recursively**

To print a BinaryTree...
- Print the root of the tree
- Print the left of the tree (recursively)
- Print the right of the tree (recursively)

```java
public void print() {
    print(this.root);
}

public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```
```java
public void print(Node c) {
  if (c != null) {
    System.out.print(c.data + " ... + "");
    print(c.left);
    print(c.right);
  }
}
```
Printing a Tree Example

```java
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```

**Post-Order Traversal**

```java
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```

**In-Order Traversal**

```java
public void print(Node c) {
    if (c != null) {
        print(c.left);
        System.out.print(c.data + " ");
        print(c.right);
    }
}
```

**Pre-Order Traversal**

```java
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```
Tree Traversal Example

Consider the following binary tree:

```
    10
   /   \
  22    3
 /  \
4   53
```

Compute the Pre-Order, In-Order, and Post-Order Traversals:
- Pre-Order: 10, 22, 3, 4, 53, 66, 17, 12
- In-Order: 3, 22, 4, 10, 66, 53, 17, 12
- Post-Order: 3, 4, 22, 66, 12, 17, 53, 10

Traversal Trick

To Quickly Generate A Traversal
- Trace a path around the tree
- As you pass a node on the proper side, process it:
  - Pre-Order: left
  - In-Order: bottom
  - Post-Order: right

Binary Tree method

Binary Tree methods are just normal recursive functions. The base case/recursive calls will always be similar.

Writing a Binary Tree Method
- The base case is current == null.
- First recursive case is method(current.left)
- Second recursive case is method(current.right)

```
1 public type method(...) {
2     return method(this.root, ...);
3 }
4 private type method(TreeNode current, ...) {
5     if (current == null) { /* DO BASE CASE */ }
6     // Do the left recursive case:
7     type leftResult = method(current.left, ...);
8     // Do the right recursive case:
9     type rightResult = method(current.right, ...);
10    /* Use the left and right results... */
11    return ...;
12 }
```

Binary Tree contains()

Write a method, in the IntTree class, called contains():
```
public boolean contains(int value) {
    return contains(this.root, value);
}
private boolean contains(IntTreeNode current, int value) {
    if (current == null) { return false; }
    else if (current.data == value) { return true; }
    else {
        boolean leftContainsValue = contains(current.left, value);
        boolean rightContainsValue = contains(current.right, value);
        return leftContainsValue || rightContainsValue;
    }
}
```

Some Tree Tips!

- Trees are just generalized LinkedLists. So, all of the things you learned about references with LinkedLists are going to apply to trees as well
- Almost all the tree methods you write will be recursive (and will have a private helper that takes in the root)
- Make sure you understand all the traversals; the trick can be very useful.