CSE 143: Computer Programming II

Outline

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

Back To LinkedLists

Consider the following standard LinkedList:

```
front
```

Recall the definition of a ListNode:

```
public class Node {
    public int data;
    public Node next;
    public Node(int data, Node next) {
        this.data = data;
        this.next = next;
    }
}
```

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

Back To LinkedLists

Nodes with Multiple next Fields

```
public class Node {
    public int data;
    public Node next1;
    public Node next2;
    public Node(int data, Node next1, Node next2) {
        this.data = data;
        this.next1 = next1;
        this.next2 = next2;
    }
}
```

Introducing Trees

Binary Trees

```
public class Node {
    public int data;
    public Node left;
    public Node right;
    public Node(int data, Node left, Node right) {
        this.data = data;
        this.left = left;
        this.right = right;
    }
}
```

(root

leaf

leaf

leaf

(red is right; yellow is left)
Consider the following LinkedList of a mathematical expression:

\[
(2 \times 4) + (7 - 3)
\]

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?

\[
+ \times \ 2 \ 4 \ - \ 7 \ 3
\]

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

More Uses of Trees

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

Printing a LinkedList Recursively

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

```
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)

```
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);
    }
}
```
Printing a Tree Example

Printing a Tree Example

Printing a Tree Example

Printing a Tree Example

Printing a Tree Example

Printing a Tree Example

Printing a Tree Example

Printing a Tree Example
Printing a Tree Example 16
1 public void print(Node c) { // c = 1
2 if (c != null) {
3 System.out.print(c.data + " ");
4 print(c.left);
5 print(c.right);
6 }
7 }

Printing a Tree Example 17
1 public void print(Node c) { // c = 1
2 if (c != null) {
3 System.out.print(c.data + " ");
4 print(c.left);
5 print(c.right);
6 }
7 }

Printing a Tree Example 18
1 public void print(Node c) { // c = 1
2 if (c != null) {
3 System.out.print(c.data + " ");
4 print(c.left);
5 print(c.right);
6 }
7 }

Printing a Tree Example 19
1 public void print(Node c) { // c = 1
2 if (c != null) {
3 System.out.print(c.data + " ");
4 print(c.left);
5 print(c.right);
6 }
7 }

Printing a Tree Example 20
1 public void print(Node c) { // c = 1
2 if (c != null) {
3 System.out.print(c.data + " ");
4 print(c.left);
5 print(c.right);
6 }
7 }

Printing a Tree Example 21
1 public void print(Node c) { // c = 1
2 if (c != null) {
3 System.out.print(c.data + " ");
4 print(c.left);
5 print(c.right);
6 }
7 }

Trace
>> 1 2 3

Trace
>> 1 2 3 4

Trace
>> 1 2 3

Trace
>> 1 2 3 4

Trace
>> 1 2 3 4 5

Trace
>> 1 2 3 4
Printing a Tree Example 22
1 public void print(Node c) { // c = 1
2 if (c != null) {
3 System.out.print(c.data + " ");
4 print(c.left);
5 print(c.right);
6 }
7 }

Trace
>> 1 2 3 4 5 6

Printing a Tree Example 23
1 public void print(Node c) { // c = 1
2 if (c != null) {
3 System.out.print(c.data + " ");
4 print(c.left);
5 print(c.right);
6 }
7 }

Trace
>> 1 2 3 4 5 6

Printing a Tree Example 24
1 public void print(Node c) { // c = 1
2 if (c != null) {
3 System.out.print(c.data + " ... + " ");
4 print(c.left);
5 print(c.right);
6 }
7 }

Trace
>> 1 2 3 4 5 6

Printing a Tree Example 25
1 public void print(Node c) { // c = 1
2 if (c != null) {
3 System.out.print(c.data + " ");
4 print(c.left);
5 print(c.right);
6 }
7 }

Trace
>> 1 2 3 4 5 6

Tree Traversals 27

Pre-Order Traversal
1 public void print(Node c) {
2 if (c != null) {
3 System.out.print(c.data + " "); // print
4 print(c.left); // left
5 print(c.right); // right
6 }
7 }

Trace
>> 1 2 3 4 5 6

In-Order Traversal
1 public void print(Node c) {
2 if (c != null) {
3 print(c.left); // left
4 System.out.print(c.data + " "); // print
5 print(c.right); // right
6 }
7 }

Trace
>> 1 2 3 4 5 6

Post-Order Traversal
1 public void print(Node c) {
2 if (c != null) {
3 print(c.left); // left
4 print(c.right); // right
5 System.out.print(c.data + " "); // print
6 }
7 }

Trace
>> 1 2 3 4 5 6
Tree Traversal Example

Consider the following binary tree:

```
root
  / \  
22   53
  /   /  
3    66
```

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.

```
public type method(...) {
    return method(this.root, ...);
}
private type method(TreeNode current, ...) {
    if (current == null) {
        /* DO BASE CASE */
    } else if (current.data == value) {
        return true;
    } else {
        boolean leftContainsValue = contains(current.left, value);
        boolean rightContainsValue = contains(current.right, value);
        return leftContainsValue || rightContainsValue;
    }
}
```

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 the tree is null, it definitely doesn't contain 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.