CSE 143

Computer Programming II

Outline

1. More Tree Methods
2. Introducing BSTs
3. BST Methods

Another Tree Method

1. height

Write a tree method called height (inside the IntTree class) with the following method signature:

```java
public int height()
```

that returns the number of nodes on the longest path from the root to any leaf. For example,

![Binary Search Trees (BSTs)](image)

height is 1  height is 5  height is 3

height Solution

```java
public int height() {
    return height(this.root);
}

private int height(IntTreeNode current) {
    // A null tree has height 0
    if (current == null) {
        return 0;
    }
    // Find the largest path by taking the max
    // of both branches recursively (and adding 1 for this node)
    return 1 + Math.max(height(current.left),
                        height(current.right));
}
```

Back to contains

Recall contains()

```java
private boolean contains(IntTreeNode current, int value) {
    /* If the tree is null, it definitely doesn’t contain value... */
    if (current == null) { return false; }
    /* If current *is* value, we found it! */
    else if (current.data == value) { return true; }
    else { return contains(current.left, value) ||
            contains(current.right, value); }
}
```

Runtime of contains(7)

Consider the following tree: Which nodes do we visit for contains(7)?

![Tree Diagram](image)

That makes the code \(O(n)\). Can we do better?
Doing Better!

In general, we can’t do better. BUT, sometimes, we can!

Definition (Binary SEARCH Tree (BST))

A binary tree is a BST when an in-order traversal of the tree yields a sorted list.

To put it another way, a binary tree is a BST when:
- All data “to the left of” a node is less than it
- All data “to the right of” a node is greater than it
- All sub-trees of the binary tree are also BSTs

Example (Which of the following are BSTs?)

This is the same tree, but now we have to visit all the nodes!

Tracing the new contains

Runtime of (better) contains(7)

Consider the following tree: Which nodes do we visit for contains(7)

That makes the code logn. Much better!

WARNING!

Consider the following tree:

This is the same tree, but now we have to visit all the nodes!

Adding to a BST (Attempt #1)

Attempt #1

What’s wrong with this solution?

Just like with LinkedLists where we must change front or .next, we’re not actually changing anything here. We’re discarding the result.

Adding to a BST (Try Again)

Write add() for a BST

Fix contains so that it takes advantage of the BST properties.

Recall contains()

That makes the code logn. Much better!

Example (tree.add(49))

Before After

Example (Which of the following are BSTs?)

NO YES NO

x = change(x)

Consider the following code:

What’s wrong with this solution?

Just like with LinkedLists where we must change front or .next, we’re not actually changing anything here. We’re discarding the result.

We must USE the result; otherwise, it gets discarded
If you want to write a method that can change the object that a variable refers to, you must do three things:

1. Pass in the original state of the object to the method
2. Return the new (possibly changed) object from the method
3. Re-assign the caller's variable to store the returned result

```java
p = change(p); // in main
```

```java
public static Point change(Point thePoint) {
    thePoint = new Point(99, -1);
    return thePoint;
}
```

This works because we always update the result, always return the result, and always update the root.

---

**BST Tips!**

- BSTs can make searching/inserting/etc. much faster.

- Make sure that you can figure out if a tree is a BST or not.

- Whenever you are writing a BST method, you **must** use the \( x = \text{change}(x) \) pattern. It won’t work otherwise.