1 (https://www.interviewcake.com/question/javascript/balanced-binary-tree)

Write a function to see if a binary tree is *"superbalanced"* (a new tree property we just made up).

A tree is "superbalanced" if the difference between the depths of any two leaf nodes is no greater than one.

Here's a sample binary tree node class:

```javascript
function BinaryTreeNode(value) {
    this.value = value;
    this.left  = null;
    this.right = null;
}

BinaryTreeNode.prototype.insertLeft = function(value) {
    this.left = new BinaryTreeNode(value);
    return this.left;
};

BinaryTreeNode.prototype.insertRight = function(value) {
    this.right = new BinaryTreeNode(value);
    return this.right;
};
```

Gotchas

Your first thought might be to write a recursive function, thinking, "the tree is balanced if the left subtree is balanced and the right subtree is balanced." This kind of approach works well for some other tree problems.

But this isn't quite true. Counterexample: suppose that from the root of our tree:

- The left subtree only has leaves at depths 10 and 11.
- The right subtree only has leaves at depths 11 and 12.

Both subtrees are balanced, but from the root we will have leaves at 3 different depths.

We could instead have our recursive function get the array of distinct leaf depths for each subtree. That could work fine. But let's come up with an iterative solution instead. It's usually better to use an iterative solution instead of a recursive one because it avoids stack overflow .

We can do this in $O(n)$ time and $O(n)$ space.

Breakdown

Sometimes it's good to start by rephrasing or "simplifying" the problem.

The requirement of "the difference between the depths of any two leaf nodes is no greater than 1" implies that we'll have to compare the depths of *all possible pairs* of leaves. That'd be expensive—if there are n leaves, there are n^2 possible pairs of leaves.

But we can simplify this requirement to require less work. For example, we could equivalently say:

- "The difference between the min leaf depth and the max leaf depth is 1 or less"
- "There are at most two distinct leaf depths, and they are at most 1 apart"

If you're having trouble with a recursive approach, try using an iterative one.

To get to our leaves and measure their depths, we'll have to traverse the tree somehow. **What methods do we know for traversing a tree?**

Depth-first and breadth-first are common ways to traverse a tree. Which one should we use here?

The worst-case time and space costs of both are the same—you could make a case for either.

But one characteristic of our algorithm is that it could **short-circuit** and return `false` as soon as it finds two leaves with depths more than 1 apart. So maybe we should **use a traversal that will hit leaves as quickly as possible...**

Depth-first traversal will generally hit leaves before breadth-first, so let's go with that. How could we write a depth-first walk that also keeps track of our depth?

Solution

We do a depth-first walk through our tree, keeping track of the depth as we go. When we find a leaf, we throw its depth into an array of `depths` *if* we haven't seen that depth already.

Each time we hit a leaf with a new depth, there are two ways that our tree might now be unbalanced:

1. There are more than 2 different leaf depths
2. There are exactly 2 leaf depths and they are more than 1 apart.

Why are we doing a depth-first walk and not a breadth-first one? You could make a case for either. We chose depth-first because it reaches leaves faster, which allows us to short-circuit earlier in some cases.

JavaScript

```
function isBalanced(treeRoot) {
    var depths = []; // we short-circuit as soon as we find more than 2

    // nodes will store pairs of a node and the node's depth
    var nodes = [];
    nodes.push([treeRoot, 0]);

    while (nodes.length) {

        // pop a node and its depth from the top of our stack
        var nodePair = nodes.pop();
        var node  = nodePair[0],
            depth = nodePair[1];

        // case: we found a leaf
        if (!node.left && !node.right) {

            // we only care if it's a new depth
            if (depths.indexOf(depth) < 0) {
                depths.push(depth);

                // two ways we might now have an unbalanced tree:
                //   1) more than 2 different leaf depths
                //   2) 2 leaf depths that are more than 1 apart
                if ((depths.length > 2) ||
                        (depths.length === 2 && Math.abs(depths[0] - depths[1]) > 1)) {
                    return false;
                }
            }

        // case: this isn't a leaf - keep stepping down
        } else {
            if (node.left) {
                nodes.push([node.left, depth + 1]);
            }
            if (node.right) {
                nodes.push([node.right, depth + 1]);
            }
        }
    }

    return true;
}
```

Type code!

Complexity

$O(n)$ time and $O(n)$ space. The `depths` array never gets bigger than 3 items so it takes $O(1)$ space, but the `nodes` stack takes $O(n)$ space.

Since the number of items in the `nodes` stack will never be greater than the height of the tree, we *could* say we're taking $O(h)$ space, where h is the height of the tree. But if our tree is just one straight line (the worst case), $h = n$.

Did you get it right?

Yes, I'm expert on this | Not quite, review later

Like this problem? Pass it on!

Share (https://www.facebook.com/sharer/sharer.php?u=https://www.interviewcake.com/question/javascript/balanced-binary-tree) Tweet (https://twitter.com/intent/tweet?text=Solved%20this%20coding%20interview%20question%21&via=interviewcake&related=binary-tree)

Yo, follow along!

(https://www.facebook.com/interviewcake) (https://twitter.com/interviewcake)

Subscribe to our weekly question email list » (/free-weekly-coding-interview-problem-newsletter)

Programming interview questions by company:

- Google interview questions (/google-interview-questions)
- Facebook interview questions (/facebook-interview-questions)
- Amazon interview questions (/amazon-interview-questions)

Programming interview questions by language:

- Java interview questions (/java-interview-questions)
- Python interview questions (/python-interview-questions)
- Ruby interview questions (/ruby-interview-questions)
- JavaScript interview questions (/javascript-interview-questions)
- **NEW:** Testing and QA interview questions (/testing-and-qa-interview-questions)
- **NEW:** SQL interview questions (/sql-interview-questions)