JavaScript Diameter of Binary Tree

Challenge

Given the `root` of a binary tree, return the length of the diameter of the tree.

The diameter of a binary tree is the length of the longest path between any two nodes in a tree. This path may or may not pass through the root.

The length of a path between two nodes is represented by the number of edges between them.

1st Example

```
Input: root = [1,2,3,4,5]
Output: 3
Explanation: 3 is the length of the path
             [4,2,1,3] or [5,2,1,3].
```

2nd Example

```
Input: root = [1,2]
Output: 1
```

Constraints

- `-100 <= Node.val <= 100`
- The number of nodes in the tree is in the range `[1, 10⁴]`.

Solution

```
const diameterOfBinaryTree = (root) => {
    let diameter = 0;

    const depth = (root) => {
        if (!root) return 0;

        let left  = depth(root.left),
            right = depth(root.right);

        diameter = Math.max(diameter, left + right);

        return longerSide = Math.max(left, right) + 1;
    };

    depth(root);

    return diameter;
};
```

Explanation

I've coded a function that calculates the diameter of a binary tree. The diameter of a binary tree is defined as the longest path between any two nodes in the tree.

The `diameterOfBinaryTree` function takes the root of the binary tree as input. It initializes the `diameter` variable to `0`.

Inside the `diameterOfBinaryTree` function, there is a nested function called `depth`. This function calculates the depth of a node in the binary tree.

Within the `depth` function, it first checks if the current node is null. If it is, it returns `0`.

Next, the `depth` function recursively calls itself on the left and right child of the current node and stores the results in the `left` and `right` variables.

The `diameter` variable is then updated by taking the maximum of its current value and the sum of `left` and `right`. This allows it to keep track of the longest path encountered so far.

Finally, the `depth` function returns the maximum of `left` and `right` plus `1`, which represents the depth of the current node.

After defining the `depth` function, the `depth` function is called with the root of the binary tree to calculate the depth of each node.

The final value of the `diameter` variable is returned as the result of the `diameterOfBinaryTree` function, representing the diameter of the binary tree.

In summary, this function uses a recursive approach to calculate the diameter of a binary tree. It defines a nested function called `depth` to calculate the depth of each node, and updates the `diameter` variable to keep track of the longest path encountered. The final diameter of the binary tree is returned as the output of the function.