JavaScript Longest Univalue Path

Challenge

Given the `root` of a binary tree, return the length of the longest path, where each node in the path has the same value. This path may or may not pass through the root.

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

1st Example

```
Input: root = [5,4,5,1,1,null,5]
Output: 2
Explanation: The shown image shows that the longest
             path of the same value (ex. 5).
```

2nd Example

```
Input: root = [1,4,5,4,4,null,5]
Output: 2
Explanation: The shown image shows that the longest
             path of the same value (ex. 4).
```

Constraints

- `-1000 <= Node.val <= 1000`
- The number of nodes in the tree is in the range `[0, 10⁴]`.
- The depth of the tree will not exceed `1000`.

Solution

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

    let max = 0;

    dfs(root);

    return max;

    function dfs(root) {
        if (!root) return 0;

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

        if (!root.left ||
            root.left.val !== root.val) left = 0;

        if (!root.right ||
            root.right.val !== root.val) right = 0;

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

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

Explanation

I've built a function called `longestUnivaluePath` that takes a binary tree as input and returns the length of the longest path in the tree where all nodes have the same value.

Inside the function, the first line checks if the root node is null. If it is null, it means the tree is empty, so the function returns `0`.

A variable called `max` is initialized to `0`. This variable will keep track of the maximum length of the univalue path.

It then calls a helper function called `dfs` passing the root node as an argument.

The `dfs` function is defined inside the main function and takes the current node as an argument.

Inside the `dfs` function, the first line checks if the current node is null. If it is null, it means we have reached the end of a branch, so the function returns `0`.

It then recursively calls the `dfs` function on the left and right children of the current node and assigns the returned values to the variables `left` and `right` respectively.

The next two lines check if the current node has a left child and if the value of the left child is different from the value of the current node. If either condition is true, it means the left child does not belong to the same univalue path, so the `left` variable is set to `0`.

Similarly, the next two lines check if the current node has a right child and if the value of the right child is different from the value of the current node. If either condition is true, it means the right child does not belong to the same univalue path, so the `right` variable is set to `0`.

It then calculates the sum of `left` and `right` and compares it

with the current maximum `max` using the `Math.max()` function. If the sum is greater than `max`, the `max` variable is updated with the new value.

Finally, it returns the maximum value between `left` and `right` incremented by `1`. This represents the length of the univalue path from the current node.

After the `dfs` function is defined, the main function returns the value of `max`, which represents the length of the longest univalue path in the binary tree.

In summary, this function uses a depth-first search (DFS) approach to traverse a binary tree and find the longest path where all nodes have the same value. It recursively calculates the length of the univalue path for each node and keeps track of the maximum length encountered. The final result is the length of the longest univalue path in the tree.

Author: Trevor Morin