Longest Univalue Path

Given a binary tree, find 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.

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

Example 1:

Input:

```
              5
             / \
            4   5
           / \   \
          1   1   5
```

Output:

```
2
```

Example 2:

Input:

```
              1
             / \
            4   5
           / \   \
          4   4   5
```

Output:

```
2
```

Note: The given binary tree has not more than 10000 nodes. The height of the tree is not more than 1000.

Solution 1

`Longest-Univalue-Path` of a tree is among those `Longest-Univalue-Path-Across` at each node;
`Longest-Univalue-Path-Across` a node is sum of { `Longest-Univalue-Path-Start-At` each child with same value, + 1}

Java

```java
class Solution {
    public int longestUnivaluePath(TreeNode root) {
        int[] res = new int[1];
        if (root != null) dfs(root, res);
        return res[0];
    }

    private int dfs(TreeNode node, int[] res) {
        int l = node.left != null ? dfs(node.left, res) : 0;
        int r = node.right != null ? dfs(node.right, res) : 0;
        int resl = node.left != null && node.left.val == node.val ? l + 1 : 0;
        int resr = node.right != null && node.right.val == node.val ? r + 1 : 0;
        res[0] = Math.max(res[0], resl + resr);
        return Math.max(resl, resr);
    }
}
```

C++

```cpp
class Solution {
public:
    int longestUnivaluePath(TreeNode* root) {
        int lup = 0;
        if (root) dfs(root, lup);
        return lup;
    }

private:
    int dfs(TreeNode* node, int& lup) {
        int l = node->left ? dfs(node->left, lup) : 0;
        int r = node->right ? dfs(node->right, lup) : 0;
        int resl = node->left && node->left->val == node->val ? l + 1 : 0;
        int resr = node->right && node->right->val == node->val ? r + 1 : 0;
        lup = max(lup, resl + resr);
        return max(resl, resr);
    }
};
```

Varables

`l` is the length of single direction `Longest-Univalue-Path` start from `left-child`,
`r` is the length of single direction `Longest-Univalue-Path` start from `right-child`,
`resl` is the length of single direction `Longest-Univalue-Path` start from `parent` go left,

`resr` is the length of single direction `Longest-Univalue-Path` start from `parent` go right.
`int dfs(node)` returns the `Longest-Univalue-Path-Start-At` that `node`, and update the result of `Longest-Univalue-Path-Across` that `node` through side effect.
It is really hard to name those variables to reflect these concept.

Example:

```
                ...
               /
              4 (res = resl + resr = 3)
  (resl = 2) / \ (resr= 1)
    (l = 1) 4   4 (r = 0)
           /
          4
```

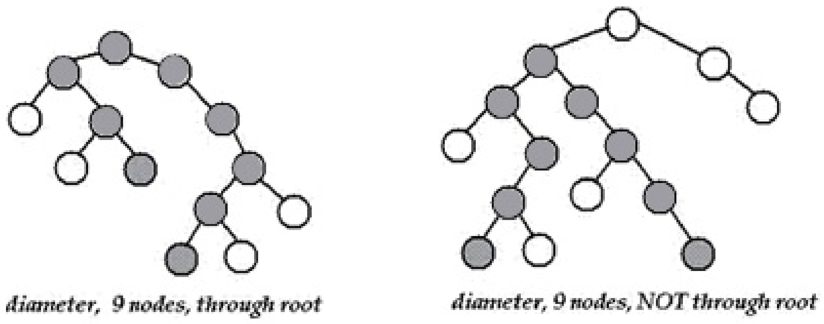
resl is `Longest-Univalue-Path-Start-At` left node + 1,
resr is `Longest-Univalue-Path-Start-At` right node + 1,
in here the local result of `Longest-Univalue-Path-Across` at this node is the sum of the 2;

written by alexander original link here

Solution 2

The approach is similar to the Diameter of Binary Tree question except that we reset the left/right to 0 whenever the current node does not match the children node value.

In the Diameter of Binary Tree question, the path can either go through the root or it doesn't.


diameter, 9 nodes, through root
diameter, 9 nodes, NOT through root

Hence at the end of each recursive loop, return the longest length using that node as the root so that the node's parent can potentially use it in its longest path computation.

We also use an external variable `longest` that keeps track of the longest path seen so far.

By Yang Shun

```
class Solution(object):
    def longestUnivaluePath(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        # Time: O(n)
        # Space: O(n)
        longest = [0]
        def traverse(node):
            if not node:
                return 0
            left_len, right_len = traverse(node.left), traverse(node.right)
            left = (left_len + 1) if node.left and node.left.val == node.val else 0
            right = (right_len + 1) if node.right and node.right.val == node.val el
se 0
            longest[0] = max(longest[0], left + right)
            return max(left, right)
        traverse(root)
        return longest[0]
```

written by yangshun original link here

Solution 3

```
class Solution {
public:
    int helper(TreeNode* root, int val)
    {
        if(!root || root->val != val) return 0;
        return 1 + max(helper(root->left,val),helper(root->right,val));
    }
    int longestUnivaluePath(TreeNode* root) {
        if(!root) return 0;
        int sub = max(longestUnivaluePath(root->left),longestUnivaluePath(root->righ
t));
        return max(sub,helper(root->left,root->val) + helper(root->right,root->val)
);
    }
};
```

written by i9r0k original link here