Height Of Binary Tree After Subtree Removal Queries

solved by	Senan
Platform	LeetCode
difficulty	Hard
# Serial	2458
tags	DFS Hash Map
language	C++
solved on	@26/10/2024
link	https://leetcode.com/problems/height-of-binary-tree-after-subtree-removal-queries/description/
Completion	☑

Intuition

To answer height-related queries for nodes in a binary tree after reversing subtrees, we can traverse the tree, keeping track of node heights and updating the max height. Using a depth-first search (DFS), we ensure that the nodes are swapped correctly and update heights based on subtree depth.

Approach

1. **DFS Traversal**: Perform a DFS traversal to keep track of the heights of each node. For each node, calculate its height and store the max height at its value in the `heights` array.
2. **Swap Subtrees**: For every node during the DFS traversal, swap its left and right children to "mirror" the tree.
3. **Handle Multiple Queries**: Run the DFS traversal twice (once before and once after mirroring the tree) to account for all possible height values in the `heights` array.
4. **Answer Queries**: For each query, simply retrieve the stored height from the `heights` array.

Complexity

Time Complexity:

The DFS traversal runs in **O(N)**, where **N** is the number of nodes in the tree. This is done twice, resulting in **O(2N)**, which simplifies to **O(N)**.

The query handling is **O(Q)**, where **Q** is the number of queries.

Overall time complexity is **O(N + Q)**.

Space Complexity:

We use an auxiliary `heights` array of size **O(100001)**, which is a constant size array for storing heights, and an `answer` array of size **O(Q)** for storing the output of queries.

Overall space complexity is **O(1)** for the `heights` array (constant) and **O(Q)** for the query results.

Code

```
class Solution {
    void dfs(TreeNode* root, vector<int> &heights, int height, int &maxHeight){
        if (root == NULL) return;

        int val = root->val;
        heights[val] = max(heights[val], height);
        maxHeight = max(height, maxHeight);

        dfs(root->left, heights, height + 1, maxHeight);
        dfs(root->right, heights, height + 1, maxHeight);

        swap(root->left, root->right);
    }

public:
    vector<int> treeQueries(TreeNode* root, vector<int>& queries) {
        vector<int> heights(100001, 0);
        int maxHeight = 0;

        dfs(root, heights, 0, maxHeight);

        maxHeight = 0;
        dfs(root, heights, 0, maxHeight);

        vector<int> answer;
        for (int elem : queries) {
            answer.push_back(heights[elem]);
        }

        return answer;
    }
};
```