Find Duplicate Subtrees

Given a binary tree, return all duplicate subtrees. For each kind of duplicate subtrees, you only need to return the root node of any **one** of them.

Two trees are duplicate if they have the same structure with same node values.

Example 1:

```
        1
       / \
      2   3
     /   / \
    4   2   4
       /
      4
```

The following are two duplicate subtrees:

```
      2
     /
    4
```

and

```
    4
```

Therefore, you need to return above trees' root in the form of a list.

Solution 1

```
public List<TreeNode> findDuplicateSubtrees(TreeNode root) {
    List<TreeNode> res = new LinkedList<>();
    postorder(root, new HashMap<>(), res);
    return res;
}

public String postorder(TreeNode cur, Map<String, Integer> map, List<TreeNode> res)
{
    if (cur == null) return "#";
    String serial = cur.val + "," + postorder(cur.left, map, res) + "," + postorder
(cur.right, map, res);
    if (map.getOrDefault(serial, 0) == 1) res.add(cur);
    map.put(serial, map.getOrDefault(serial, 0) + 1);
    return serial;
}
```

written by compton_scatter original link here

Solution 2

```
class Solution {
public:
    vector<TreeNode*> findDuplicateSubtrees(TreeNode* root) {
        unordered_map<string, vector<TreeNode*>> map;
        vector<TreeNode*> dups;
        serialize(root, map);
        for (auto it = map.begin(); it != map.end(); it++) {
            if (it->second.size() > 1) {
                dups.push_back(it->second[0]);
            }
        }

        return dups;
    }

private:
    string serialize(TreeNode* node, unordered_map<string, vector<TreeNode*>>& map)
{
        if (!node) return "";
        string s = "(" + serialize(node->left, map) + to_string(node->val) + serial
ize(node->right, map) + ")";
        map[s].push_back(node);
        return s;
    }
};
```

written by alexander original link here

Solution 3

```
def findDuplicateSubtrees(self, root):
    def tuplify(root):
        if root:
            tuple = root.val, tuplify(root.left), tuplify(root.right)
            trees[tuple].append(root)
            return tuple
    trees = collections.defaultdict(list)
    tuplify(root)
    return [roots[0] for roots in trees.values() if roots[1:]]
```

I convert the entire tree of nested `TreeNode`s to a tree of nested `tuple`s. Those have the advantage that they already support hashing and deep comparison (for the very unlikely cases of hash collisions). So then I can just use each subtree's `tuple` version as a key in my dictionary. And equal subtrees have the same key and thus get collected in the same list.

Overall this costs only O(n) memory (where n is the number of nodes in the given tree). The string serialization I've seen in other posted solutions costs O(n^2) memory (and thus also at least that much time).

written by StefanPochmann original link here