Check if a given Binary Tree is SumTree

February 22, 2018

Question

Write a function that returns true if the given Binary Tree is SumTree else false. A SumTree is a Binary Tree where the value of a node is equal to sum of the nodes present in its left subtree and right subtree. An empty tree is SumTree and sum of an empty tree can be considered as 0. A leaf node is also considered as SumTree.

Explanation and Algorithm

One approach would be to get the sum of the nodes in the right and left subtree.Check if the calculated sum equals the root's data. Recursively check if each subtree is a SumTree.

Another more tricky approach would be to determine if the node is a leaf node or not. If it is, then sum of subtree rooted with this node is equal to value of this node. Otherwise, the sum of subtree rooted with this node is twice the value of this node (Assuming that the tree rooted with this node is SumTree).

Hints

1. Think of a base case for a recursive method, involving the smallest binary sum tree. Apply the same logic to a bigger tree. Think tree within tree.
2. Make sure you understand the definition of a SumTree. All the subtrees have to be Sumtrees too, not just the main tree.

Code

```
/*Recursive Approach*/
   /* returns 1 if sum property holds for the given
       node and both of its children */
    int isSumTree(Node node){
```

```
        int ls;
        int rs;

        /* If node is NULL or it's a leaf node then
           return true */
        if ((node == null) || (node.left == null && node.right == null))
            return 1;

        /* Get sum of nodes in left and right subtrees */
        ls = sum(node.left);
        rs = sum(node.right);

        /* if the node and both of its children satisfy the
           property return 1 else 0*/
        if ((node.data == ls + rs) && (isSumTree(node.left) != 0)&&
            (isSumTree(node.right)) != 0){
            return 1;
      }
        return 0;
    }
```

```
/*Leaf Node Approach*/
   /* returns 1 if sum property holds for the given
      node and both of its children */
    int isSumTree(Node node){
        int ls;
        int rs;

        /* If node is NULL or it's a leaf node then
           return true */
        if ((node == null) || (node.left == null && node.right == null))
            return 1;

        /* Get sum of nodes in left and right subtrees */
        ls = sum(node.left);
        rs = sum(node.right);

        /* if the node and both of its children satisfy the
           property return 1 else 0*/
        if ((node.data == ls + rs) && (isSumTree(node.left) != 0)&&
            (isSumTree(node.right)) != 0){
            return 1;
      }
        return 0;
    }
```

```
 /* returns 1 if SumTree property holds for the given tree */
    int isSumTree(Node node) {
```

```
    int ls; // for sum of nodes in left subtree
    int rs; // for sum of nodes in right subtree

    /* If node is NULL or it's a leaf node then
     return true */
    if (node == null || isLeaf(node) == 1)
        return 1;

    if (isSumTree(node.left) != 0 && isSumTree(node.right) != 0) {
        // Get the sum of nodes in left subtree
        if (node.left == null)
            ls = 0;
        else if (isLeaf(node.left) != 0)
            ls = node.left.data;
        else
            ls = 2 * (node.left.data);

        // Get the sum of nodes in right subtree
        if (node.right == null)
            rs = 0;
        else if (isLeaf(node.right) != 0)
            rs = node.right.data;
        else
            rs = 2 * (node.right.data);

        /* If root's data is equal to sum of nodes in left
           and right subtrees then return 1 else return 0*/
        if ((node.data == rs + ls))
            return 1;
        else
            return 0;
    }

    return 0;
}
```

Run time Analysis

For the first strategy, the worst case would be $O(n^2)$.
For the second, the run time is $O(n)$.