Find Count of Single Valued Subtrees

Given a binary tree, write a program to count the number of Single Valued Subtrees. A Single Valued Subtree is one in which all the nodes have same value. Expected time complexity is O(n).

Example:

```
Input: root of below tree
              5
             / \
            1   5
           / \   \
          5   5   5
Output: 4
There are 4 subtrees with single values.


Input: root of below tree
              5
             / \
            4   5
           / \   \
          4   4   5
Output: 5
There are five subtrees with single values.
```

We strongly recommend you to minimize your browser and try this yourself first.

A **Simple Solution** is to traverse the tree. For every traversed node, check if all values under this node are same or not. If same, then increment count. Time complexity of this solution is $O(n^2)$.

An **Efficient Solution** is to traverse the tree in bottom up manner. For every subtree visited, return true if subtree rooted under it is single valued and increment count. So the idea is to use count as a reference parameter in recursive calls and use returned values to find out if left and right subtrees are single valued or not.

Below is the implementation of above idea.

C++

```
// C++ program to find count of single valued subtrees
#include<bits/stdc++.h>
using namespace std;

// A Tree node
struct Node
{
    int data;
    struct Node* left, *right;
};

// Utility function to create a new node
Node* newNode(int data)
{
    Node* temp = new Node;
    temp->data = data;
    temp->left = temp->right = NULL;
    return (temp);
}

// This function increments count by number of single
// valued subtrees under root. It returns true if subtree
// under root is Singly, else false.
bool countSingleRec(Node* root, int &count)
```

```
{
    // Return false to indicate NULL
    if (root == NULL)
       return true;

    // Recursively count in left and right subtrees also
    bool left = countSingleRec(root->left, count);
    bool right = countSingleRec(root->right, count);

    // If any of the subtrees is not singly, then this
    // cannot be singly.
    if (left == false || right == false)
       return false;

    // If left subtree is singly and non-empty, but data
    // doesn't match
    if (root->left && root->data != root->left->data)
        return false;

    // Same for right subtree
    if (root->right && root->data != root->right->data)
        return false;

    // If none of the above conditions is true, then
    // tree rooted under root is single valued, increment
    // count and return true.
    count++;
    return true;
}

// This function mainly calls countSingleRec()
// after initializing count as 0
int countSingle(Node* root)
{
    // Initialize result
    int count = 0;

    // Recursive function to count
    countSingleRec(root, count);

    return count;
}

// Driver program to test
int main()
{
    /* Let us construct the below tree
            5
          /   \
        4      5
      /  \      \
     4    4      5 */
    Node* root        = newNode(5);
    root->left        = newNode(4);
    root->right       = newNode(5);
    root->left->left  = newNode(4);
    root->left->right = newNode(4);
    root->right->right = newNode(5);

    cout << "Count of Single Valued Subtrees is "
         << countSingle(root);
    return 0;
}
```

Java

```
// Java program to find count of single valued subtrees

/* Class containing left and right child of current
 node and key value*/
```

```
class Node
{
    int data;
    Node left, right;

    public Node(int item)
    {
        data = item;
        left = right = null;
    }
}

class Count
{
    int count = 0;
}

class BinaryTree
{
    Node root;
    Count ct = new Count();

    // This function increments count by number of single
    // valued subtrees under root. It returns true if subtree
    // under root is Singly, else false.
    boolean countSingleRec(Node node, Count c)
    {
        // Return false to indicate NULL
        if (node == null)
            return true;

        // Recursively count in left and right subtrees also
        boolean left = countSingleRec(node.left, c);
        boolean right = countSingleRec(node.right, c);

        // If any of the subtrees is not singly, then this
        // cannot be singly.
        if (left == false || right == false)
            return false;

        // If left subtree is singly and non-empty, but data
        // doesn't match
        if (node.left != null && node.data != node.left.data)
            return false;

        // Same for right subtree
        if (node.right != null && node.data != node.right.data)
            return false;

        // If none of the above conditions is true, then
        // tree rooted under root is single valued, increment
        // count and return true.
        c.count++;
        return true;
    }

    // This function mainly calls countSingleRec()
    // after initializing count as 0
    int countSingle()
    {
        return countSingle(root);
    }

    int countSingle(Node node)
    {
        // Recursive function to count
        countSingleRec(node, ct);
        return ct.count;
    }

    // Driver program to test above functions
    public static void main(String args[])
    {
```

```
    {
           /* Let us construct the below tree
                5
              /   \
            4      5
          /  \      \
         4    4      5 */
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(5);
        tree.root.left = new Node(4);
        tree.root.right = new Node(5);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(4);
        tree.root.right.right = new Node(5);

        System.out.println("The count of single valued sub trees is : "
                                           + tree.countSingle());
    }
}

// This code has been contributed by Mayank Jaiswal
```

Python

```
# Python program to find the count of single valued subtrees

# Node Structure
class Node:
    # Utility function to create a new node
    def __init__(self ,data):
        self.data = data
        self.left = None
        self.right = None


# This function increments count by number of single
# valued subtrees under root. It returns true if subtree
# under root is Singly, else false.
def countSingleRec(root , count):
    # Return False to indicate None
    if root is None :
        return True

    # Recursively count in left and right subtress also
    left = countSingleRec(root.left , count)
    right = countSingleRec(root.right , count)

    # If any of the subtress is not singly, then this
    # cannot be singly
    if left == False or right  == False :
        return False

    # If left subtree is singly and non-empty , but data
    # doesn't match
    if root.left and root.data != root.left.data:
        return False

    # same for right subtree
    if root.right and root.data != root.right.data:
        return False

    # If none of the above conditions is True, then
    # tree rooted under root is single valued,increment
    # count and return true
    count[0] += 1
    return True


# This function mainly calss countSingleRec()
# after initializing count as 0
def countSingle(root):
```

```
    # initialize result
    count = [0]

    # Recursive function to count
    countSingleRec(root , count)

    return count[0]


# Driver program to test

"""Let us contruct the below tree
            5
          /   \
        4       5
       /  \      \
      4    4      5
"""
root = Node(5)
root.left = Node(4)
root.right = Node(5)
root.left.left = Node(4)
root.left.right = Node(4)
root.right.right = Node(5)
countSingle(root)
print "Count of Single Valued Subtress is" , countSingle(root)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Output:

```
Count of Single Valued Subtrees is 5
```

Time complexity of this solution is O(n) where n is number of nodes in given binary tree.