Problem 250: Count Univalue Subtrees

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Given the

root

of a binary tree, return

the number of
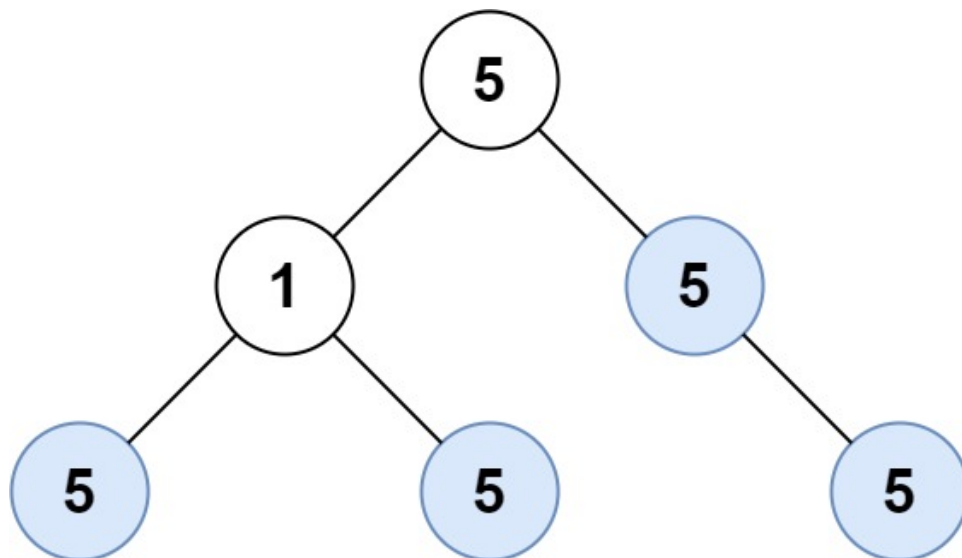
uni-value

subtrees

.

A

uni-value subtree

means all nodes of the subtree have the same value.

Example 1:

Input:

root = [5,1,5,5,5,null,5]

Output:

4

Example 2:

Input:

root = []

Output:

0

Example 3:

Input:

root = [5,5,5,5,5,null,5]

Output:

Constraints:

The number of the node in the tree will be in the range

[0, 1000]

.

-1000 <= Node.val <= 1000

Code Snippets

C++:

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode() : val(0), left(nullptr), right(nullptr) {}
 * TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 * TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
int countUnivalSubtrees(TreeNode* root) {

}
};
```

Java:

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
```

```
 *  int val;
 *  TreeNode left;
 *  TreeNode right;
 *  TreeNode() {}
 *  TreeNode(int val) { this.val = val; }
 *  TreeNode(int val, TreeNode left, TreeNode right) {
 *  this.val = val;
 *  this.left = left;
 *  this.right = right;
 *  }
 *  }
 */
class Solution {
public int countUnivalSubtrees(TreeNode root) {

}
}
```

Python3:

```
# Definition for a binary tree node.
# class TreeNode:
# def __init__(self, val=0, left=None, right=None):
# self.val = val
# self.left = left
# self.right = right
class Solution:
def countUnivalSubtrees(self, root: Optional[TreeNode]) -> int:
```

Python:

```
# Definition for a binary tree node.
# class TreeNode(object):
# def __init__(self, val=0, left=None, right=None):
# self.val = val
# self.left = left
# self.right = right
class Solution(object):
def countUnivalSubtrees(self, root):
"""
:type root: Optional[TreeNode]
:rtype: int
```

```
"""
```

JavaScript:

```javascript
/**
 * Definition for a binary tree node.
 * function TreeNode(val, left, right) {
 * this.val = (val===undefined ? 0 : val)
 * this.left = (left===undefined ? null : left)
 * this.right = (right===undefined ? null : right)
 * }
 */
/**
 * @param {TreeNode} root
 * @return {number}
 */
var countUnivalSubtrees = function(root) {

};
```

TypeScript:

```typescript
/**
 * Definition for a binary tree node.
 * class TreeNode {
 * val: number
 * left: TreeNode | null
 * right: TreeNode | null
 * constructor(val?: number, left?: TreeNode | null, right?: TreeNode | null)
 {
 * this.val = (val===undefined ? 0 : val)
 * this.left = (left===undefined ? null : left)
 * this.right = (right===undefined ? null : right)
 * }
 * }
 */

function countUnivalSubtrees(root: TreeNode | null): number {

};
```

C#:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 * public int val;
 * public TreeNode left;
 * public TreeNode right;
 * public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
 * this.val = val;
 * this.left = left;
 * this.right = right;
 * }
 * }
 */
public class Solution {
public int CountUnivalSubtrees(TreeNode root) {


}
}
```

C:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * struct TreeNode *left;
 * struct TreeNode *right;
 * };
 */
int countUnivalSubtrees(struct TreeNode* root) {


}
```

Go:

```
/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 * Val int
 * Left *TreeNode
 * Right *TreeNode
 * }
 */
```

```
func countUnivalSubtrees(root *TreeNode) int {


}
```

Kotlin:

```
/**
 * Example:
 * var ti = TreeNode(5)
 * var v = ti.`val`
 * Definition for a binary tree node.
 * class TreeNode(var `val`: Int) {
 * var left: TreeNode? = null
 * var right: TreeNode? = null
 * }
 */
class Solution {
fun countUnivalSubtrees(root: TreeNode?): Int {


}
}
```

Swift:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 * public var val: Int
 * public var left: TreeNode?
 * public var right: TreeNode?
 * public init() { self.val = 0; self.left = nil; self.right = nil; }
 * public init(_ val: Int) { self.val = val; self.left = nil; self.right =
nil; }
 * public init(_ val: Int, _ left: TreeNode?, _ right: TreeNode?) {
 * self.val = val
 * self.left = left
 * self.right = right
 * }
 * }
 */
class Solution {
func countUnivalSubtrees(_ root: TreeNode?) -> Int {
```

```
        }
    }
```

Rust:

```rust
// Definition for a binary tree node.
// #[derive(Debug, PartialEq, Eq)]
// pub struct TreeNode {
// pub val: i32,
// pub left: Option<Rc<RefCell<TreeNode>>>,
// pub right: Option<Rc<RefCell<TreeNode>>>,
// }
//
// impl TreeNode {
// #[inline]
// pub fn new(val: i32) -> Self {
// TreeNode {
// val,
// left: None,
// right: None
// }
// }
// }
use std::rc::Rc;
use std::cell::RefCell;
impl Solution {
pub fn count_unival_subtrees(root: Option<Rc<RefCell<TreeNode>>>) -> i32 {


}
}
```

Ruby:

```ruby
# Definition for a binary tree node.
# class TreeNode
# attr_accessor :val, :left, :right
# def initialize(val = 0, left = nil, right = nil)
# @val = val
# @left = left
# @right = right
# end
```

```ruby
  # end
  # @param {TreeNode} root
  # @return {Integer}
  def count_unival_subtrees(root)

  end
```

PHP:

```php
/**
 * Definition for a binary tree node.
 * class TreeNode {
 *     public $val = null;
 *     public $left = null;
 *     public $right = null;
 *     function __construct($val = 0, $left = null, $right = null) {
 *         $this->val = $val;
 *         $this->left = $left;
 *         $this->right = $right;
 *     }
 * }
 */
class Solution {

    /**
     * @param TreeNode $root
     * @return Integer
     */
    function countUnivalSubtrees($root) {

    }
}
```

Dart:

```dart
/**
 * Definition for a binary tree node.
 * class TreeNode {
 *     int val;
 *     TreeNode? left;
 *     TreeNode? right;
 *     TreeNode([this.val = 0, this.left, this.right]);
```

```
* }
*/
class Solution {
int countUnivalSubtrees(TreeNode? root) {

}
}
```

Scala:

```
/**
 * Definition for a binary tree node.
 * class TreeNode(_value: Int = 0, _left: TreeNode = null, _right: TreeNode =
 null) {
 * var value: Int = _value
 * var left: TreeNode = _left
 * var right: TreeNode = _right
 * }
 */
object Solution {
def countUnivalSubtrees(root: TreeNode): Int = {

}
}
```

Elixir:

```
# Definition for a binary tree node.
#
# defmodule TreeNode do
# @type t :: %__MODULE__{
# val: integer,
# left: TreeNode.t() | nil,
# right: TreeNode.t() | nil
# }
# defstruct val: 0, left: nil, right: nil
# end

defmodule Solution do
@spec count_unival_subtrees(root :: TreeNode.t | nil) :: integer
def count_unival_subtrees(root) do
```

```
        end
    end
```

Erlang:

```erlang
%% Definition for a binary tree node.
%%
%% -record(tree_node, {val = 0 :: integer(),
%% left = null :: 'null' | #tree_node{},
%% right = null :: 'null' | #tree_node{}}).

-spec count_unival_subtrees(Root :: #tree_node{} | null) -> integer().
count_unival_subtrees(Root) ->
    .
```

Racket:

```racket
; Definition for a binary tree node.
#|

; val : integer?
; left : (or/c tree-node? #f)
; right : (or/c tree-node? #f)
(struct tree-node
(val left right) #:mutable #:transparent)

; constructor
(define (make-tree-node [val 0])
(tree-node val #f #f))

|#

(define/contract (count-unival-subtrees root)
(-> (or/c tree-node? #f) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Univalue Subtrees
* Difficulty: Medium
* Tags: tree, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/


/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* TreeNode *left;
* TreeNode *right;
* TreeNode() : val(0), left(nullptr), right(nullptr) {
// TODO: Implement optimized solution
return 0;
}
* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {
// TODO: Implement optimized solution
return 0;
}
* TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {
// TODO: Implement optimized solution
return 0;
}
* };
*/
class Solution {
public:
int countUnivalSubtrees(TreeNode* root) {

}
};
```

Java Solution:

```
/**
* Problem: Count Univalue Subtrees
```

```
 * Difficulty: Medium
 * Tags: tree, search
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(h) for recursion stack where h is height
 */


/**
 * Definition for a binary tree node.
 * public class TreeNode {
 * int val;
 * TreeNode left;
 * TreeNode right;
 * TreeNode() {
// TODO: Implement optimized solution
return 0;
}
 * TreeNode(int val) { this.val = val; }
 * TreeNode(int val, TreeNode left, TreeNode right) {
 * this.val = val;
 * this.left = left;
 * this.right = right;
 * }
 * }
 */
class Solution {
public int countUnivalSubtrees(TreeNode root) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Univalue Subtrees
Difficulty: Medium
Tags: tree, search


Approach: DFS or BFS traversal
Time Complexity: O(n) where n is number of nodes
```

```
    Space Complexity: O(h) for recursion stack where h is height
    """


    # Definition for a binary tree node.
    # class TreeNode:
    # def __init__(self, val=0, left=None, right=None):
    # self.val = val
    # self.left = left
    # self.right = right
    class Solution:
    def countUnivalSubtrees(self, root: Optional[TreeNode]) -> int:
    # TODO: Implement optimized solution
    pass
```

Python Solution:

```python
    # Definition for a binary tree node.
    # class TreeNode(object):
    # def __init__(self, val=0, left=None, right=None):
    # self.val = val
    # self.left = left
    # self.right = right
    class Solution(object):
    def countUnivalSubtrees(self, root):
    """
    :type root: Optional[TreeNode]
    :rtype: int
    """
```

JavaScript Solution:

```javascript
    /**
    * Problem: Count Univalue Subtrees
    * Difficulty: Medium
    * Tags: tree, search
    *
    * Approach: DFS or BFS traversal
    * Time Complexity: O(n) where n is number of nodes
    * Space Complexity: O(h) for recursion stack where h is height
    */
```

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val, left, right) {
 * this.val = (val===undefined ? 0 : val)
 * this.left = (left===undefined ? null : left)
 * this.right = (right===undefined ? null : right)
 * }
 */
/**
 * @param {TreeNode} root
 * @return {number}
 */
var countUnivalSubtrees = function(root) {

};
```

TypeScript Solution:

```
/**
 * Problem: Count Univalue Subtrees
 * Difficulty: Medium
 * Tags: tree, search
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(h) for recursion stack where h is height
 */


/**
 * Definition for a binary tree node.
 * class TreeNode {
 * val: number
 * left: TreeNode | null
 * right: TreeNode | null
 * constructor(val?: number, left?: TreeNode | null, right?: TreeNode | null)
 {
 * this.val = (val===undefined ? 0 : val)
 * this.left = (left===undefined ? null : left)
 * this.right = (right===undefined ? null : right)
 * }
 * }
```

```
*/

function countUnivalSubtrees(root: TreeNode | null): number {

};
```

C# Solution:

```
/*
* Problem: Count Univalue Subtrees
* Difficulty: Medium
* Tags: tree, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Definition for a binary tree node.
* public class TreeNode {
* public int val;
* public TreeNode left;
* public TreeNode right;
* public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
*/
public class Solution {
public int CountUnivalSubtrees(TreeNode root) {

}
}
```

C Solution:

```
/*
* Problem: Count Univalue Subtrees
```

```
 * Difficulty: Medium
 * Tags: tree, search
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(h) for recursion stack where h is height
 */


/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * struct TreeNode *left;
 * struct TreeNode *right;
 * };
 */
int countUnivalSubtrees(struct TreeNode* root) {


}
```

Go Solution:

```go
// Problem: Count Univalue Subtrees
// Difficulty: Medium
// Tags: tree, search
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 * Val int
 * Left *TreeNode
 * Right *TreeNode
 * }
 */
func countUnivalSubtrees(root *TreeNode) int {


}
```

Kotlin Solution:

```kotlin
/**
 * Example:
 * var ti = TreeNode(5)
 * var v = ti.`val`
 * Definition for a binary tree node.
 * class TreeNode(var `val`: Int) {
 * var left: TreeNode? = null
 * var right: TreeNode? = null
 * }
 */
class Solution {
fun countUnivalSubtrees(root: TreeNode?): Int {


}
}
```

Swift Solution:

```swift
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 * public var val: Int
 * public var left: TreeNode?
 * public var right: TreeNode?
 * public init() { self.val = 0; self.left = nil; self.right = nil; }
 * public init(_ val: Int) { self.val = val; self.left = nil; self.right =
nil; }
 * public init(_ val: Int, _ left: TreeNode?, _ right: TreeNode?) {
 * self.val = val
 * self.left = left
 * self.right = right
 * }
 * }
 */
class Solution {
func countUnivalSubtrees(_ root: TreeNode?) -> Int {


}
}
```

Rust Solution:

```rust
// Problem: Count Univalue Subtrees
// Difficulty: Medium
// Tags: tree, search
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

// Definition for a binary tree node.
// #[derive(Debug, PartialEq, Eq)]
// pub struct TreeNode {
//   pub val: i32,
//   pub left: Option<Rc<RefCell<TreeNode>>>,
//   pub right: Option<Rc<RefCell<TreeNode>>>,
// }
//
// impl TreeNode {
//   #[inline]
//   pub fn new(val: i32) -> Self {
//     TreeNode {
//       val,
//       left: None,
//       right: None
//     }
//   }
// }
use std::rc::Rc;
use std::cell::RefCell;
impl Solution {
pub fn count_unival_subtrees(root: Option<Rc<RefCell<TreeNode>>>) -> i32 {

}
}
```

Ruby Solution:

```ruby
# Definition for a binary tree node.
# class TreeNode
#   attr_accessor :val, :left, :right
#   def initialize(val = 0, left = nil, right = nil)
#     @val = val
```

```
# @left = left
# @right = right
# end
# end
# @param {TreeNode} root
# @return {Integer}
def count_unival_subtrees(root)


end
```

PHP Solution:

```php
/**
 * Definition for a binary tree node.
 * class TreeNode {
 * public $val = null;
 * public $left = null;
 * public $right = null;
 * function __construct($val = 0, $left = null, $right = null) {
 * $this->val = $val;
 * $this->left = $left;
 * $this->right = $right;
 * }
 * }
 */
class Solution {

/**
 * @param TreeNode $root
 * @return Integer
 */
function countUnivalSubtrees($root) {


}
}
```

Dart Solution:

```dart
/**
 * Definition for a binary tree node.
 * class TreeNode {
```

```
 * int val;
 * TreeNode? left;
 * TreeNode? right;
 * TreeNode([this.val = 0, this.left, this.right]);
 * }
 */
class Solution {
int countUnivalSubtrees(TreeNode? root) {


}
}
```

Scala Solution:

```
/**
 * Definition for a binary tree node.
 * class TreeNode(_value: Int = 0, _left: TreeNode = null, _right: TreeNode =
null) {
 * var value: Int = _value
 * var left: TreeNode = _left
 * var right: TreeNode = _right
 * }
 */
object Solution {
def countUnivalSubtrees(root: TreeNode): Int = {


}
}
```

Elixir Solution:

```
# Definition for a binary tree node.
#
# defmodule TreeNode do
# @type t :: %__MODULE__{
# val: integer,
# left: TreeNode.t() | nil,
# right: TreeNode.t() | nil
# }
# defstruct val: 0, left: nil, right: nil
# end
```

```
defmodule Solution do
@spec count_unival_subtrees(root :: TreeNode.t | nil) :: integer
def count_unival_subtrees(root) do

end
end
```

Erlang Solution:

```
%% Definition for a binary tree node.
%%
%% -record(tree_node, {val = 0 :: integer(),
%% left = null :: 'null' | #tree_node{},
%% right = null :: 'null' | #tree_node{}}).

-spec count_unival_subtrees(Root :: #tree_node{} | null) -> integer().
count_unival_subtrees(Root) ->
  .
```

Racket Solution:

```
; Definition for a binary tree node.
#|

; val : integer?
; left : (or/c tree-node? #f)
; right : (or/c tree-node? #f)
(struct tree-node
(val left right) #:mutable #:transparent)

; constructor
(define (make-tree-node [val 0])
(tree-node val #f #f))

|#

(define/contract (count-unival-subtrees root)
(-> (or/c tree-node? #f) exact-integer?)
)
```