Problem 515: Find Largest Value in Each Tree Row

Problem Information

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

Problem Description

Given the

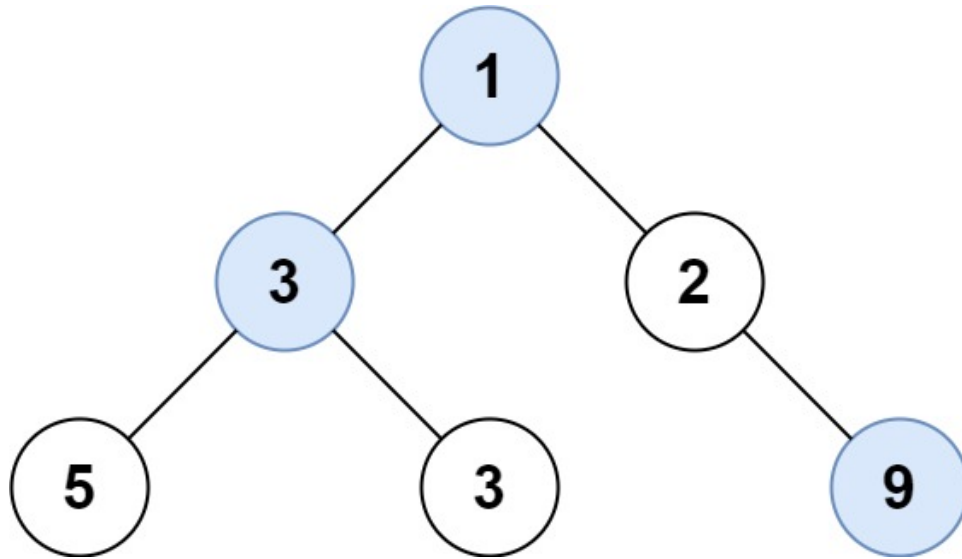root

of a binary tree, return

an array of the largest value in each row

of the tree

(0-indexed)

.

Example 1:

Input:

root = [1,3,2,5,3,null,9]

Output:

[1,3,9]

Example 2:

Input:

root = [1,2,3]

Output:

[1,3]

Constraints:

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

[0, 10

4

]

.

-2

31

<= Node.val <= 2

31

- 1

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:
vector<int> largestValues(TreeNode* root) {

}
};
```

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 List<Integer> largestValues(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 largestValues(self, root: Optional[TreeNode]) -> List[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 largestValues(self, root):
"""
```

```
:type root: Optional[TreeNode]
:rtype: List[int]
"""
```

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 largestValues = function(root) {

};
```

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 largestValues(root: TreeNode | null): number[] {

};
```

C#:

```csharp
/**
 * 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 IList<int> LargestValues(TreeNode root) {

}
}
```

C:

```c
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * struct TreeNode *left;
 * struct TreeNode *right;
 * };
 */
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* largestValues(struct TreeNode* root, int* returnSize) {

}
```

Go:

```go
/**
 * Definition for a binary tree node.
 * type TreeNode struct {
```

```
 * Val int
 * Left *TreeNode
 * Right *TreeNode
 * }
 */
func largestValues(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 largestValues(root: TreeNode?): List<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 largestValues(_ 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 largest_values(root: Option<Rc<RefCell<TreeNode>>>) -> Vec<i32> {


}
}
```

Ruby:

```ruby
# Definition for a binary tree node.
# class TreeNode
# attr_accessor :val, :left, :right
```

```ruby
# def initialize(val = 0, left = nil, right = nil)
#     @val = val
#     @left = left
#     @right = right
# end
# end
# @param {TreeNode} root
# @return {Integer[]}
def largest_values(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 $root
     * @return Integer[]
     */
    function largestValues($root) {

    }
}
```

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 {
List<int> largestValues(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 largestValues(root: TreeNode): List[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 largest_values(root :: TreeNode.t | nil) :: [integer]
def largest_values(root) do

end
end
```

Erlang:

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

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

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 (largest-values root)
(-> (or/c tree-node? #f) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```cpp
/*
 * Problem: Find Largest Value in Each Tree Row
 * Difficulty: Medium
 * Tags: array, tree, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * 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) {}
 * 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:
vector<int> largestValues(TreeNode* root) {

}
};
```

Java Solution:

```java
/**
 * Problem: Find Largest Value in Each Tree Row
 * Difficulty: Medium
 * Tags: array, tree, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * 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() {}
 * 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 List<Integer> largestValues(TreeNode root) {


}
}
```

Python3 Solution:

```
"""
Problem: Find Largest Value in Each Tree Row
Difficulty: Medium
Tags: array, tree, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
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
```

```python
class Solution:
def largestValues(self, root: Optional[TreeNode]) -> List[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 largestValues(self, root):
"""
:type root: Optional[TreeNode]
:rtype: List[int]
"""
```

JavaScript Solution:

```javascript
/**
 * Problem: Find Largest Value in Each Tree Row
 * Difficulty: Medium
 * Tags: array, tree, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * 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 largestValues = function(root) {

};
```

TypeScript Solution:

```typescript
/**
 * Problem: Find Largest Value in Each Tree Row
 * Difficulty: Medium
 * Tags: array, tree, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * 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 largestValues(root: TreeNode | null): number[] {

};
```

C# Solution:

```
/*
* Problem: Find Largest Value in Each Tree Row
* Difficulty: Medium
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* 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 IList<int> LargestValues(TreeNode root) {

}
}
```

C Solution:

```
/*
* Problem: Find Largest Value in Each Tree Row
* Difficulty: Medium
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* 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;
* };
*/
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* largestValues(struct TreeNode* root, int* returnSize) {


}
```

Go Solution:

```go
// Problem: Find Largest Value in Each Tree Row
// Difficulty: Medium
// Tags: array, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// 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 largestValues(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 largestValues(root: TreeNode?): List<Int> {


}
}
```

Swift Solution:

```
/**
 * 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 largestValues(_ root: TreeNode?) -> [Int] {


}
}
```

Rust Solution:

```
// Problem: Find Largest Value in Each Tree Row
// Difficulty: Medium
```

```rust
// Tags: array, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// 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 largest_values(root: Option<Rc<RefCell<TreeNode>>>) -> Vec<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 largest_values(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 $root
 * @return Integer[]
 */
function largestValues($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 {
List<int> largestValues(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 largestValues(root: TreeNode): List[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 largest_values(root :: TreeNode.t | nil) :: [integer]
def largest_values(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 largest_values(Root :: #tree_node{} | null) -> [integer()].
largest_values(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 (largest-values root)
(-> (or/c tree-node? #f) (listof exact-integer?))
)
```