Problem 814: Binary Tree Pruning

Problem Information

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

Problem Description

Given the

root

of a binary tree, return

the same tree where every subtree (of the given tree) not containing a

1

has been removed

.

A subtree of a node

node
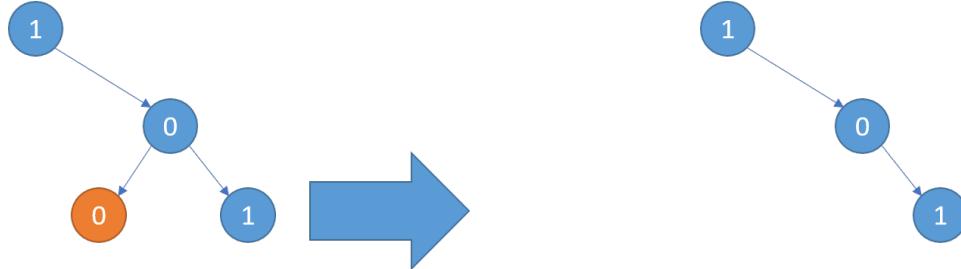
is

node

plus every node that is a descendant of

node

.

Example 1:

Input:

root = [1,null,0,0,1]
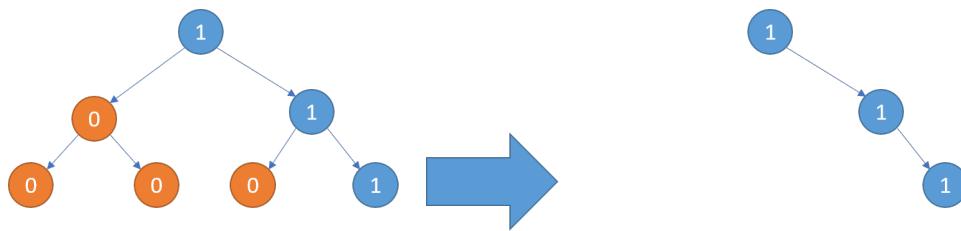
Output:

[1,null,0,null,1]

Explanation:

Only the red nodes satisfy the property "every subtree not containing a 1". The diagram on the right represents the answer.

Example 2:
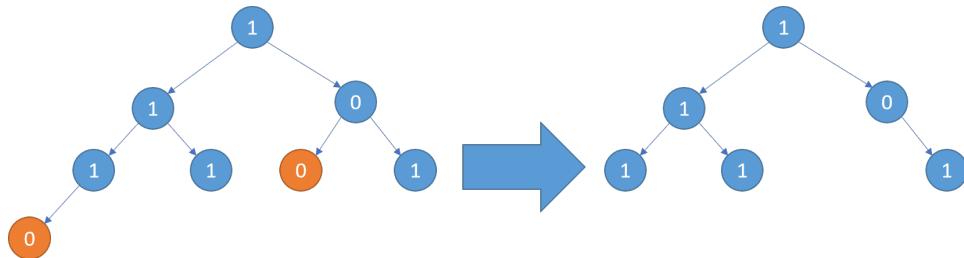
Input:

root = [1,0,1,0,0,0,1]

Output:

[1,null,1,null,1]

Example 3:

Input:

root = [1,1,0,1,1,0,1,0]

Output:

[1,1,0,1,1,null,1]

Constraints:

The number of nodes in the tree is in the range

[1, 200]

.

Node.val

is either

0

or

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:
TreeNode* pruneTree(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 TreeNode pruneTree(TreeNode root) {


}
}
```

Python3:

```python
# 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 pruneTree(self, root: Optional[TreeNode]) -> Optional[TreeNode]:
```

Python:

```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 pruneTree(self, root):
"""
:type root: Optional[TreeNode]
:rtype: Optional[TreeNode]
"""
```

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 {TreeNode}
 */
var pruneTree = 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 pruneTree(root: TreeNode | null): TreeNode | null {

};
```

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 TreeNode PruneTree(TreeNode root) {

}
}
```

C:

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


}
```

Go:

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


}
```

Kotlin:

```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 pruneTree(root: TreeNode?): TreeNode? {
```

```
        }
    }
```

Swift:

```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 pruneTree(_ root: TreeNode?) -> TreeNode? {


    }
}
```

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,
```

```rust
//      left: None,
//      right: None
//    }
//  }
// }
use std::rc::Rc;
use std::cell::RefCell;
impl Solution {
pub fn prune_tree(root: Option<Rc<RefCell<TreeNode>>>) ->
Option<Rc<RefCell<TreeNode>>> {


}
}
```

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
# end
# @param {TreeNode} root
# @return {TreeNode}
def prune_tree(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 $root
    * @return TreeNode
    */
    function pruneTree($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 {
    TreeNode? pruneTree(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 pruneTree(root: TreeNode): TreeNode = {


}
}
```

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 prune_tree(root :: TreeNode.t | nil) :: TreeNode.t | nil
def prune_tree(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 prune_tree(Root :: #tree_node{} | null) -> #tree_node{} | null.
prune_tree(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 (prune-tree root)
(-> (or/c tree-node? #f) (or/c tree-node? #f))
)
```

Solutions

C++ Solution:

```cpp
/*
 * Problem: Binary Tree Pruning
 * 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) {}
 * 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:
TreeNode* pruneTree(TreeNode* root) {


}
};
```

Java Solution:

```
/**
 * Problem: Binary Tree Pruning
 * 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 TreeNode pruneTree(TreeNode root) {

}
}
```

Python3 Solution:

```
"""
Problem: Binary Tree Pruning
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 pruneTree(self, root: Optional[TreeNode]) -> Optional[TreeNode]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
# 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 pruneTree(self, root):
"""
:type root: Optional[TreeNode]
```

```
        :rtype: Optional[TreeNode]
        """
```

JavaScript Solution:

```javascript
/**
 * Problem: Binary Tree Pruning
 * 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 {TreeNode}
 */
var pruneTree = function(root) {

};
```

TypeScript Solution:

```typescript
/**
 * Problem: Binary Tree Pruning
 * 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 pruneTree(root: TreeNode | null): TreeNode | null {

};
```

C# Solution:

```
/*
 * Problem: Binary Tree Pruning
 * 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 TreeNode PruneTree(TreeNode root) {


}
}
```

C Solution:

```
/*
* Problem: Binary Tree Pruning
* 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;
* };
*/
struct TreeNode* pruneTree(struct TreeNode* root) {


}
```

Go Solution:

```
// Problem: Binary Tree Pruning
// 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 pruneTree(root *TreeNode) *TreeNode {


}
```

Kotlin Solution:

```
/**
* 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 pruneTree(root: TreeNode?): TreeNode? {


}
}
```

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 pruneTree(_ root: TreeNode?) -> TreeNode? {

}
}
```

Rust Solution:

```
// Problem: Binary Tree Pruning
// 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,
```

```rust
// right: None
// }
// }
// }
use std::rc::Rc;
use std::cell::RefCell;
impl Solution {
pub fn prune_tree(root: Option<Rc<RefCell<TreeNode>>>) ->
Option<Rc<RefCell<TreeNode>>> {


}
}
```

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 {TreeNode}
def prune_tree(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 $root
 * @return TreeNode
 */
function pruneTree($root) {


}
}
```

Dart Solution:

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


}
}
```

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 prune_tree(root :: TreeNode.t | nil) :: TreeNode.t | nil
def prune_tree(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 prune_tree(Root :: #tree_node{} | null) -> #tree_node{} | null.
prune_tree(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 (prune-tree root)
(-> (or/c tree-node? #f) (or/c tree-node? #f))
)
```