Problem 1339: Maximum Product of Splitted Binary Tree

Problem Information

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

Problem Description

Given the

root

of a binary tree, split the binary tree into two subtrees by removing one edge such that the product of the sums of the subtrees is maximized.

Return

the maximum product of the sums of the two subtrees

. Since the answer may be too large, return it
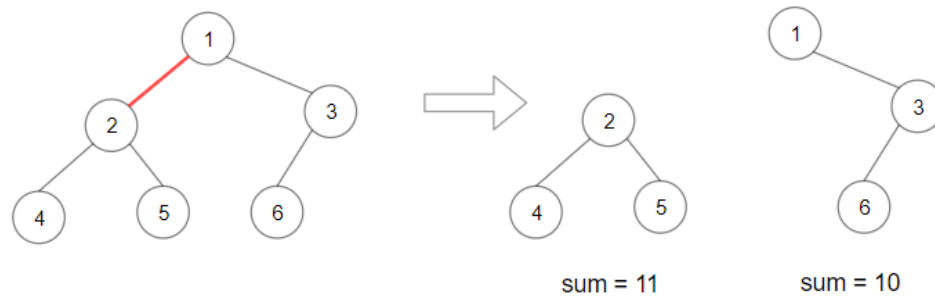
modulo

10

9

+ 7

.

Note

that you need to maximize the answer before taking the mod and not after taking it.

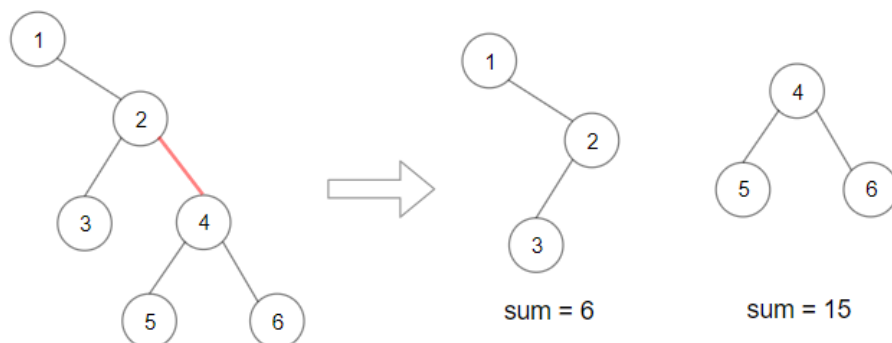Example 1:

sum = 11   sum = 10

Input:

root = [1,2,3,4,5,6]

Output:

110

Explanation:

Remove the red edge and get 2 binary trees with sum 11 and 10. Their product is 110 (11*10)

Example 2:

sum = 6   sum = 15

Input:

root = [1,null,2,3,4,null,null,5,6]

Output:

90

Explanation:

Remove the red edge and get 2 binary trees with sum 15 and 6.Their product is 90 (15*6)

Constraints:

The number of nodes in the tree is in the range

[2, 5 * 10

4

]

.

1 <= Node.val <= 10

4

Code Snippets

C++:

```
/**
 * 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 maxProduct(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 int maxProduct(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 maxProduct(self, root: Optional[TreeNode]) -> int:
```

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 maxProduct(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 maxProduct = 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 maxProduct(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 MaxProduct(TreeNode root) {

}
}
```

C:

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

```
int maxProduct(struct TreeNode* root) {


}
```

Go:

```
/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 * Val int
 * Left *TreeNode
 * Right *TreeNode
 * }
 */
func maxProduct(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 maxProduct(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 maxProduct(_ 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 max_product(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
# end
# @param {TreeNode} root
# @return {Integer}
def max_product(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 maxProduct($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 {
int maxProduct(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 maxProduct(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 max_product(root :: TreeNode.t | nil) :: integer
def max_product(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 max_product(Root :: #tree_node{} | null) -> integer().
max_product(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 (max-product root)
(-> (or/c tree-node? #f) exact-integer?)
)
```

Solutions

C++ Solution:

```cpp
/*
 * Problem: Maximum Product of Splitted Binary Tree
 * 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 maxProduct(TreeNode* root) {


    }
    };
```

Java Solution:

```java
/**
 * Problem: Maximum Product of Splitted Binary Tree
 * 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 {
```

```java
public int maxProduct(TreeNode root) {

}
}
```

Python3 Solution:

```python
"""
Problem: Maximum Product of Splitted Binary Tree
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 maxProduct(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 maxProduct(self, root):
"""
:type root: Optional[TreeNode]
:rtype: int
```

```
"""
```

JavaScript Solution:

```javascript
/**
 * Problem: Maximum Product of Splitted Binary Tree
 * 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 maxProduct = function(root) {

};
```

TypeScript Solution:

```typescript
/**
 * Problem: Maximum Product of Splitted Binary Tree
 * 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 maxProduct(root: TreeNode | null): number {

};
```

C# Solution:

```
/*
 * Problem: Maximum Product of Splitted Binary Tree
 * 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 MaxProduct(TreeNode root) {


}
}
```

C Solution:

```c
/*
* Problem: Maximum Product of Splitted Binary Tree
* 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 maxProduct(struct TreeNode* root) {


}
```

Go Solution:

```go
// Problem: Maximum Product of Splitted Binary Tree
// 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 maxProduct(root *TreeNode) int {

}
```

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 maxProduct(root: TreeNode?): 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 maxProduct(_ root: TreeNode?) -> Int {


}
}
```

Rust Solution:

```
// Problem: Maximum Product of Splitted Binary Tree
// 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
```

```rust
// }
// }
// }
use std::rc::Rc;
use std::cell::RefCell;
impl Solution {
pub fn max_product(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 max_product(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 $root
     * @return Integer
     */
    function maxProduct($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 {
  int maxProduct(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 maxProduct(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 max_product(root :: TreeNode.t | nil) :: integer
def max_product(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 max_product(Root :: #tree_node{} | null) -> integer().
max_product(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 (max-product root)
(-> (or/c tree-node? #f) exact-integer?)
)
```