Problem 1130: Minimum Cost Tree From Leaf Values

Problem Information

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

Problem Description

Given an array

arr

of positive integers, consider all binary trees such that:

Each node has either

0

or

2

children;

The values of

arr

correspond to the values of each

leaf

in an in-order traversal of the tree.

The value of each non-leaf node is equal to the product of the largest leaf value in its left and right subtree, respectively.

Among all possible binary trees considered, return

the smallest possible sum of the values of each non-leaf node

. It is guaranteed this sum fits into a

32-bit

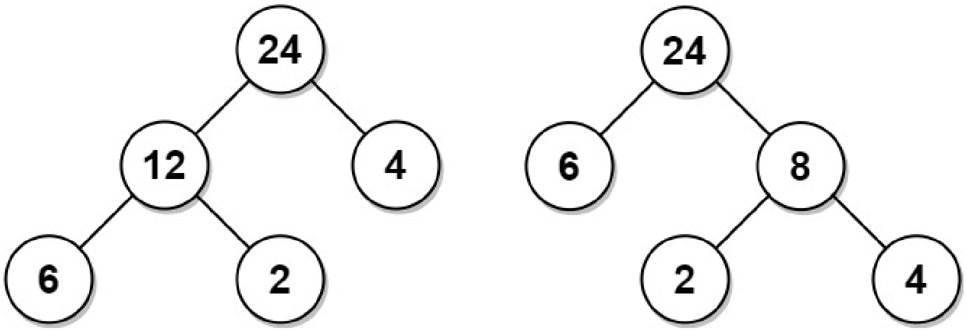
integer.

A node is a

leaf

if and only if it has zero children.

Example 1:



Input:

arr = [6,2,4]

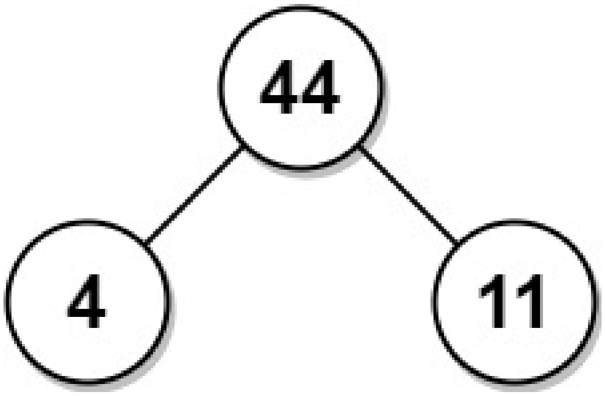
Output:

32

Explanation:

There are two possible trees shown. The first has a non-leaf node sum 36, and the second has non-leaf node sum 32.

Example 2:



Input:

arr = [4,11]

Output:

44

Constraints:

2 <= arr.length <= 40

1 <= arr[i] <= 15

It is guaranteed that the answer fits into a

32-bit

signed integer (i.e., it is less than 2

31

).

Code Snippets

C++:

```
class Solution {
public:
int mctFromLeafValues(vector<int>& arr) {

}
};
```

Java:

```
class Solution {
public int mctFromLeafValues(int[] arr) {

}
}
```

Python3:

```
class Solution:
def mctFromLeafValues(self, arr: List[int]) -> int:
```

Python:

```
class Solution(object):
def mctFromLeafValues(self, arr):
"""
:type arr: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} arr
* @return {number}
*/
var mctFromLeafValues = function(arr) {

};
```

TypeScript:

```
function mctFromLeafValues(arr: number[]): number {

};
```

C#:

```
public class Solution {
public int MctFromLeafValues(int[] arr) {

}
}
```

C:

```
int mctFromLeafValues(int* arr, int arrSize) {

}
```

Go:

```
func mctFromLeafValues(arr []int) int {

}
```

Kotlin:

```
class Solution {
fun mctFromLeafValues(arr: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func mctFromLeafValues(_ arr: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn mct_from_leaf_values(arr: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @return {Integer}
def mct_from_leaf_values(arr)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer
*/
function mctFromLeafValues($arr) {

}
}
```

Dart:

```
class Solution {
int mctFromLeafValues(List<int> arr) {

}
}
```

Scala:

```
object Solution {
def mctFromLeafValues(arr: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec mct_from_leaf_values(arr :: [integer]) :: integer
def mct_from_leaf_values(arr) do

end
end
```

Erlang:

```
-spec mct_from_leaf_values(Arr :: [integer()]) -> integer().
mct_from_leaf_values(Arr) ->
.
```

Racket:

```
(define/contract (mct-from-leaf-values arr)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Cost Tree From Leaf Values
* Difficulty: Medium
* Tags: array, tree, dp, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int mctFromLeafValues(vector<int>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Cost Tree From Leaf Values
* Difficulty: Medium
* Tags: array, tree, dp, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int mctFromLeafValues(int[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Cost Tree From Leaf Values
Difficulty: Medium
Tags: array, tree, dp, greedy, stack

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def mctFromLeafValues(self, arr: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def mctFromLeafValues(self, arr):
"""
:type arr: List[int]
:rtype: int
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Cost Tree From Leaf Values
* Difficulty: Medium
* Tags: array, tree, dp, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} arr
* @return {number}
*/
var mctFromLeafValues = function(arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Cost Tree From Leaf Values
* Difficulty: Medium
* Tags: array, tree, dp, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function mctFromLeafValues(arr: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Cost Tree From Leaf Values
* Difficulty: Medium
* Tags: array, tree, dp, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MctFromLeafValues(int[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Cost Tree From Leaf Values
* Difficulty: Medium
* Tags: array, tree, dp, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int mctFromLeafValues(int* arr, int arrSize) {

}
```

Go Solution:

```
// Problem: Minimum Cost Tree From Leaf Values
// Difficulty: Medium
// Tags: array, tree, dp, greedy, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table
```

```
func mctFromLeafValues(arr []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun mctFromLeafValues(arr: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func mctFromLeafValues(_ arr: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Cost Tree From Leaf Values
// Difficulty: Medium
// Tags: array, tree, dp, greedy, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn mct_from_leaf_values(arr: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @return {Integer}
def mct_from_leaf_values(arr)
```

```


end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer
*/
function mctFromLeafValues($arr) {

}
}
```

Dart Solution:

```
class Solution {
int mctFromLeafValues(List<int> arr) {

}
}
```

Scala Solution:

```
object Solution {
def mctFromLeafValues(arr: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec mct_from_leaf_values(arr :: [integer]) :: integer
def mct_from_leaf_values(arr) do

end
end
```

Erlang Solution:

```
-spec mct_from_leaf_values(Arr :: [integer()]) -> integer().
mct_from_leaf_values(Arr) ->
  .
```

Racket Solution:

```
(define/contract (mct-from-leaf-values arr)
  (-> (listof exact-integer?) exact-integer?)
  )
```