Problem 3117: Minimum Sum of Values by Dividing Array

Problem Information

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

Problem Description

You are given two arrays

nums

and

andValues

of length

n

and

m

respectively.

The

value

of an array is equal to the

last

element of that array.

You have to divide

nums

into

m

disjoint contiguous

subarrays

such that for the

i

th

subarray

[l

i

, r

i

]

, the bitwise

AND

of the subarray elements is equal to

andValues[i]

, in other words,

nums[l

i

] & nums[l

i

+ 1] & ... & nums[r

i

] == andValues[i]

for all

1 <= i <= m

, where

&

represents the bitwise

AND

operator.

Return

the

minimum

possible sum of the

values

of the

m

subarrays

nums

is divided into

.

If it is not possible to divide

nums

into

m

subarrays satisfying these conditions, return

-1

.

Example 1:

Input:

nums = [1,4,3,3,2], andValues = [0,3,3,2]

Output:

12

Explanation:

The only possible way to divide

nums

is:

[1,4]

as

1 & 4 == 0

.

[3]

as the bitwise

AND

of a single element subarray is that element itself.

[3]

as the bitwise

AND

of a single element subarray is that element itself.

[2]

as the bitwise

AND

of a single element subarray is that element itself.

The sum of the values for these subarrays is

4 + 3 + 3 + 2 = 12

.

Example 2:

Input:

nums = [2,3,5,7,7,7,5], andValues = [0,7,5]

Output:

17

Explanation:

There are three ways to divide

nums

:

[[2,3,5],[7,7,7],[5]]

with the sum of the values

5 + 7 + 5 == 17

.

[[2,3,5,7],[7,7],[5]]

with the sum of the values

7 + 7 + 5 == 19

.

[[2,3,5,7,7],[7],[5]]

with the sum of the values

7 + 7 + 5 == 19

.

The minimum possible sum of the values is

17

.

Example 3:

Input:

nums = [1,2,3,4], andValues = [2]

Output:

-1

Explanation:

The bitwise

AND

of the entire array

nums

is

0

. As there is no possible way to divide

nums

into a single subarray to have the bitwise

AND

of elements

2

, return

-1

.

Constraints:

1 <= n == nums.length <= 10

4

1 <= m == andValues.length <= min(n, 10)

1 <= nums[i] < 10

5

0 <= andValues[j] < 10

5

Code Snippets

C++:

```
class Solution {
public:
int minimumValueSum(vector<int>& nums, vector<int>& andValues) {

}
};
```

Java:

```
class Solution {
public int minimumValueSum(int[] nums, int[] andValues) {

}
}
```

Python3:

```
class Solution:
def minimumValueSum(self, nums: List[int], andValues: List[int]) -> int:
```

Python:

```
class Solution(object):
def minimumValueSum(self, nums, andValues):
"""
:type nums: List[int]
:type andValues: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number[]} andValues
* @return {number}
*/
var minimumValueSum = function(nums, andValues) {

};
```

TypeScript:

```
function minimumValueSum(nums: number[], andValues: number[]): number {

};
```

C#:

```
public class Solution {
public int MinimumValueSum(int[] nums, int[] andValues) {

}
}
```

C:

```
int minimumValueSum(int* nums, int numsSize, int* andValues, int
andValuesSize) {

}
```

Go:

```
func minimumValueSum(nums []int, andValues []int) int {

}
```

Kotlin:

```
class Solution {
fun minimumValueSum(nums: IntArray, andValues: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minimumValueSum(_ nums: [Int], _ andValues: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_value_sum(nums: Vec<i32>, and_values: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[]} and_values
# @return {Integer}
def minimum_value_sum(nums, and_values)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $andValues
* @return Integer
*/
function minimumValueSum($nums, $andValues) {

}
}
```

Dart:

```
class Solution {
int minimumValueSum(List<int> nums, List<int> andValues) {

}
}
```

Scala:

```
object Solution {
def minimumValueSum(nums: Array[Int], andValues: Array[Int]): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec minimum_value_sum(nums :: [integer], and_values :: [integer]) ::
integer
def minimum_value_sum(nums, and_values) do

end
end
```

Erlang:

```
-spec minimum_value_sum(Nums :: [integer()], AndValues :: [integer()]) ->
integer().
minimum_value_sum(Nums, AndValues) ->
.
```

Racket:

```
(define/contract (minimum-value-sum nums andValues)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Sum of Values by Dividing Array
* Difficulty: Hard
* Tags: array, tree, dp, search, queue
*
* 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 minimumValueSum(vector<int>& nums, vector<int>& andValues) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Sum of Values by Dividing Array
* Difficulty: Hard
* Tags: array, tree, dp, search, queue
*
* 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 minimumValueSum(int[] nums, int[] andValues) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Sum of Values by Dividing Array
Difficulty: Hard
Tags: array, tree, dp, search, queue

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 minimumValueSum(self, nums: List[int], andValues: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumValueSum(self, nums, andValues):
"""
:type nums: List[int]
:type andValues: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Sum of Values by Dividing Array
* Difficulty: Hard
* Tags: array, tree, dp, search, queue
*
* 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[]} nums
* @param {number[]} andValues
* @return {number}
*/
var minimumValueSum = function(nums, andValues) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Sum of Values by Dividing Array
* Difficulty: Hard
* Tags: array, tree, dp, search, queue
*
* 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 minimumValueSum(nums: number[], andValues: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Sum of Values by Dividing Array
* Difficulty: Hard
* Tags: array, tree, dp, search, queue
*
* 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 MinimumValueSum(int[] nums, int[] andValues) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Sum of Values by Dividing Array
* Difficulty: Hard
* Tags: array, tree, dp, search, queue
*
* 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 minimumValueSum(int* nums, int numsSize, int* andValues, int
andValuesSize) {

}
```

Go Solution:

```
// Problem: Minimum Sum of Values by Dividing Array
// Difficulty: Hard
// Tags: array, tree, dp, search, queue
//
// 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 minimumValueSum(nums []int, andValues []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumValueSum(nums: IntArray, andValues: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumValueSum(_ nums: [Int], _ andValues: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Sum of Values by Dividing Array
// Difficulty: Hard
// Tags: array, tree, dp, search, queue
//
// 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 minimum_value_sum(nums: Vec<i32>, and_values: Vec<i32>) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[]} and_values
# @return {Integer}
def minimum_value_sum(nums, and_values)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $andValues
* @return Integer
*/
function minimumValueSum($nums, $andValues) {

}
}
```

Dart Solution:

```
class Solution {
int minimumValueSum(List<int> nums, List<int> andValues) {

}
}
```

Scala Solution:

```
object Solution {
def minimumValueSum(nums: Array[Int], andValues: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_value_sum(nums :: [integer], and_values :: [integer]) ::
integer
def minimum_value_sum(nums, and_values) do

end
end
```

Erlang Solution:

```
-spec minimum_value_sum(Nums :: [integer()], AndValues :: [integer()]) ->
integer().
minimum_value_sum(Nums, AndValues) ->
.
```

Racket Solution:

```
(define/contract (minimum-value-sum nums andValues)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```