Problem 2588: Count the Number of Beautiful Subarrays

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

. In one operation, you can:

Choose two different indices

i

and

j

such that

0 <= i, j < nums.length

.

Choose a non-negative integer

k

such that the

k

th

bit (

0-indexed

) in the binary representation of

nums[i]

and

nums[j]

is

1

.

Subtract

2

k

from

nums[i]

and

nums[j]

.

A subarray is

beautiful

if it is possible to make all of its elements equal to

0

after applying the above operation any number of times (including zero).

Return

the number of

beautiful subarrays

in the array

nums

.

A subarray is a contiguous

non-empty

sequence of elements within an array.

Note

: Subarrays where all elements are initially 0 are considered beautiful, as no operation is needed.

Example 1:

Input:

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

Output:

2

Explanation:

There are 2 beautiful subarrays in nums: [4,

3,1,2

,4] and [

4,3,1,2,4

]. - We can make all elements in the subarray [3,1,2] equal to 0 in the following way: - Choose [

3

, 1,

2

] and k = 1. Subtract 2

1

from both numbers. The subarray becomes [1, 1, 0]. - Choose [

1

,

1

, 0] and k = 0. Subtract 2

0

from both numbers. The subarray becomes [0, 0, 0]. - We can make all elements in the subarray [4,3,1,2,4] equal to 0 in the following way: - Choose [

4

, 3, 1, 2,

4

] and k = 2. Subtract 2

2

from both numbers. The subarray becomes [0, 3, 1, 2, 0]. - Choose [0,

3

,

1

, 2, 0] and k = 0. Subtract 2

0

from both numbers. The subarray becomes [0, 2, 0, 2, 0]. - Choose [0,

2

, 0,

2

, 0] and k = 1. Subtract 2

1

from both numbers. The subarray becomes [0, 0, 0, 0, 0].

Example 2:

Input:

nums = [1,10,4]

Output:

0

Explanation:

There are no beautiful subarrays in nums.

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 10

6

Code Snippets

C++:

```
class Solution {
public:
long long beautifulSubarrays(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public long beautifulSubarrays(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function beautifulSubarrays(nums: number[]): number {

};
```

C#:

```
public class Solution {
public long BeautifulSubarrays(int[] nums) {
```

```
}
}
```

C:

```
long long beautifulSubarrays(int* nums, int numsSize) {

}
```

Go:

```
func beautifulSubarrays(nums []int) int64 {

}
```

Kotlin:

```
class Solution {
fun beautifulSubarrays(nums: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func beautifulSubarrays(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn beautiful_subarrays(nums: Vec<i32>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def beautiful_subarrays(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function beautifulSubarrays($nums) {

}
}
```

Dart:

```
class Solution {
int beautifulSubarrays(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def beautifulSubarrays(nums: Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec beautiful_subarrays(nums :: [integer]) :: integer
def beautiful_subarrays(nums) do

end
end
```

Erlang:

```
-spec beautiful_subarrays(Nums :: [integer()]) -> integer().
beautiful_subarrays(Nums) ->
.
```

Racket:

```
(define/contract (beautiful-subarrays nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count the Number of Beautiful Subarrays
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
long long beautifulSubarrays(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Count the Number of Beautiful Subarrays
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public long beautifulSubarrays(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Count the Number of Beautiful Subarrays
Difficulty: Medium
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count the Number of Beautiful Subarrays
* Difficulty: Medium
```

```
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} nums
* @return {number}
*/
var beautifulSubarrays = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Count the Number of Beautiful Subarrays
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function beautifulSubarrays(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count the Number of Beautiful Subarrays
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

public class Solution {
public long BeautifulSubarrays(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Count the Number of Beautiful Subarrays
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

long long beautifulSubarrays(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Count the Number of Beautiful Subarrays
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func beautifulSubarrays(nums []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun beautifulSubarrays(nums: IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func beautifulSubarrays(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count the Number of Beautiful Subarrays
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn beautiful_subarrays(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def beautiful_subarrays(nums)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $nums
* @return Integer
*/
function beautifulSubarrays($nums) {

}
}
```

Dart Solution:

```
class Solution {
int beautifulSubarrays(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def beautifulSubarrays(nums: Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec beautiful_subarrays(nums :: [integer]) :: integer
def beautiful_subarrays(nums) do

end
end
```

Erlang Solution:

```
-spec beautiful_subarrays(Nums :: [integer()]) -> integer().
beautiful_subarrays(Nums) ->
.
```

Racket Solution:

```
(define/contract (beautiful-subarrays nums)
(-> (listof exact-integer?) exact-integer?)
)
```