Problem 1994: The Number of Good Subsets

Problem Information

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

Problem Description

You are given an integer array

nums

. We call a subset of

nums

good

if its product can be represented as a product of one or more

distinct prime

numbers.

For example, if

nums = [1, 2, 3, 4]

:

[2, 3]

,

[1, 2, 3]

, and

[1, 3]

are

good

subsets with products

6 = 2*3

,

6 = 2*3

, and

3 = 3

respectively.

[1, 4]

and

[4]

are not

good

subsets with products

4 = 2*2

and

4 = 2*2

respectively.

Return

the number of different

good

subsets in

nums

modulo

10

9

+ 7

.

A

subset

of

nums

is any array that can be obtained by deleting some (possibly none or all) elements from

nums

. Two subsets are different if and only if the chosen indices to delete are different.

Example 1:

Input:

nums = [1,2,3,4]

Output:

6

Explanation:

The good subsets are: - [1,2]: product is 2, which is the product of distinct prime 2. - [1,2,3]: product is 6, which is the product of distinct primes 2 and 3. - [1,3]: product is 3, which is the product of distinct prime 3. - [2]: product is 2, which is the product of distinct prime 2. - [2,3]: product is 6, which is the product of distinct primes 2 and 3. - [3]: product is 3, which is the product of distinct prime 3.

Example 2:

Input:

nums = [4,2,3,15]

Output:

5

Explanation:

The good subsets are: - [2]: product is 2, which is the product of distinct prime 2. - [2,3]: product is 6, which is the product of distinct primes 2 and 3. - [2,15]: product is 30, which is the product of distinct primes 2, 3, and 5. - [3]: product is 3, which is the product of distinct prime 3. - [15]: product is 15, which is the product of distinct primes 3 and 5.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 30

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int numberOfGoodSubsets(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
```

```
 * @return {number}
 */
var numberOfGoodSubsets = function(nums) {

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int NumberOfGoodSubsets(int[] nums) {

}
}
```

C:

```
int numberOfGoodSubsets(int* nums, int numsSize) {

}
```

Go:

```
func numberOfGoodSubsets(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun numberOfGoodSubsets(nums: IntArray): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (number-of-good-subsets nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: The Number of Good Subsets
* Difficulty: Hard
* Tags: array, dp, math, hash
*
* 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 numberOfGoodSubsets(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: The Number of Good Subsets
* Difficulty: Hard
* Tags: array, dp, math, hash
*
* 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 numberOfGoodSubsets(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: The Number of Good Subsets
Difficulty: Hard
Tags: array, dp, math, hash

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

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: The Number of Good Subsets
* Difficulty: Hard
* Tags: array, dp, math, hash
*
* 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
* @return {number}
*/
var numberOfGoodSubsets = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: The Number of Good Subsets
* Difficulty: Hard
* Tags: array, dp, math, hash
*
* 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 numberOfGoodSubsets(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: The Number of Good Subsets
* Difficulty: Hard
* Tags: array, dp, math, hash
*
* 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 NumberOfGoodSubsets(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: The Number of Good Subsets
* Difficulty: Hard
* Tags: array, dp, math, hash
*
* 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 numberOfGoodSubsets(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: The Number of Good Subsets
// Difficulty: Hard
// Tags: array, dp, math, hash
//
// 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 numberOfGoodSubsets(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numberOfGoodSubsets(nums: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: The Number of Good Subsets
// Difficulty: Hard
// Tags: array, dp, math, hash
//
// 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 number_of_good_subsets(nums: Vec<i32>) -> i32 {

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (number-of-good-subsets nums)
(-> (listof exact-integer?) exact-integer?)
)
```