Problem 2572: Count the Number of Square-Free Subsets

Problem Information

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

Problem Description

You are given a positive integer

0-indexed

array

nums

.

A subset of the array

nums

is

square-free

if the product of its elements is a

square-free integer

.

A

square-free integer

is an integer that is divisible by no square number other than

1

.

Return

the number of square-free non-empty subsets of the array

nums

. Since the answer may be too large, return it

modulo

10

9

+ 7

.

A

non-empty

subset

of

nums

is an array that can be obtained by deleting some (possibly none but not all) elements from

nums

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

Example 1:

Input:

nums = [3,4,4,5]

Output:

3

Explanation:

There are 3 square-free subsets in this example: - The subset consisting of the 0

th

element [3]. The product of its elements is 3, which is a square-free integer. - The subset consisting of the 3

rd

element [5]. The product of its elements is 5, which is a square-free integer. - The subset consisting of 0

th

and 3

rd

elements [3,5]. The product of its elements is 15, which is a square-free integer. It can be proven that there are no more than 3 square-free subsets in the given array.

Example 2:

Input:

nums = [1]

Output:

1

Explanation:

There is 1 square-free subset in this example: - The subset consisting of the 0

th

element [1]. The product of its elements is 1, which is a square-free integer. It can be proven that there is no more than 1 square-free subset in the given array.

Constraints:

1 <= nums.length <= 1000

1 <= nums[i] <= 30

Code Snippets

C++:

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

}
};
```

Java:

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

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (square-free-subsets nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count the Number of Square-Free Subsets
* Difficulty: Medium
* Tags: array, dp, math
*
* 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 squareFreeSubsets(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Count the Number of Square-Free Subsets
* Difficulty: Medium
* Tags: array, dp, math
*
* 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 squareFreeSubsets(int[] nums) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Count the Number of Square-Free Subsets
Difficulty: Medium
Tags: array, dp, math

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count the Number of Square-Free Subsets
* Difficulty: Medium
* Tags: array, dp, math
*
* 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 squareFreeSubsets = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Count the Number of Square-Free Subsets
* Difficulty: Medium
* Tags: array, dp, math
*
* 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 squareFreeSubsets(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count the Number of Square-Free Subsets
* Difficulty: Medium
* Tags: array, dp, math
*
* 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 SquareFreeSubsets(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Count the Number of Square-Free Subsets
* Difficulty: Medium
* Tags: array, dp, math
*
* 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 squareFreeSubsets(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Count the Number of Square-Free Subsets
// Difficulty: Medium
// Tags: array, dp, math
//
// 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 squareFreeSubsets(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Count the Number of Square-Free Subsets
// Difficulty: Medium
// Tags: array, dp, math
//
// 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 square_free_subsets(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (square-free-subsets nums)
(-> (listof exact-integer?) exact-integer?)
)
```