Problem 3539: Find Sum of Array Product of Magical Sequences

Problem Information

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

Problem Description

You are given two integers,

m

and

k

, and an integer array

nums

.

A sequence of integers

seq

is called

magical

if:

seq

has a size of

m

.

0 <= seq[i] < nums.length

The

binary representation

of

2

seq[0]

+ 2

seq[1]

+ ... + 2

seq[m - 1]

has

k

set bits

.

The

array product

of this sequence is defined as

prod(seq) = (nums[seq[0]] * nums[seq[1]] * ... * nums[seq[m - 1]])

.

Return the

sum

of the

array products

for all valid

magical

sequences.

Since the answer may be large, return it

modulo

10

9

+ 7

.

A

set bit

refers to a bit in the binary representation of a number that has a value of 1.

Example 1:

Input:

m = 5, k = 5, nums = [1,10,100,10000,1000000]

Output:

991600007

Explanation:

All permutations of

[0, 1, 2, 3, 4]

are magical sequences, each with an array product of 10

13

.

Example 2:

Input:

m = 2, k = 2, nums = [5,4,3,2,1]

Output:

170

Explanation:

The magical sequences are

[0, 1]

,

[0, 2]

,

[0, 3]

,

[0, 4]

,

[1, 0]

,

[1, 2]

,

[1, 3]

,

[1, 4]

,

[2, 0]

,

[2, 1]

,

[2, 3]

,

[2, 4]

,

[3, 0]

,

[3, 1]

,

[3, 2]

,

[3, 4]

,

[4, 0]

,

[4, 1]

,

[4, 2]

, and

[4, 3]

.

Example 3:

Input:

m = 1, k = 1, nums = [28]

Output:

28

Explanation:

The only magical sequence is

[0]

.

Constraints:

1 <= k <= m <= 30

1 <= nums.length <= 50

1 <= nums[i] <= 10

8

Code Snippets

C++:

```
class Solution {
public:
int magicalSum(int m, int k, vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int magicalSum(int m, int k, int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number} m
* @param {number} k
* @param {number[]} nums
* @return {number}
*/
var magicalSum = function(m, k, nums) {

};
```

TypeScript:

```
function magicalSum(m: number, k: number, nums: number[]): number {

};
```

C#:

```
public class Solution {
public int MagicalSum(int m, int k, int[] nums) {

}
}
```

C:

```
int magicalSum(int m, int k, int* nums, int numsSize) {

}
```

Go:

```
func magicalSum(m int, k int, nums []int) int {

}
```

Kotlin:

```
class Solution {
fun magicalSum(m: Int, k: Int, nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func magicalSum(_ m: Int, _ k: Int, _ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn magical_sum(m: i32, k: i32, nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} m
# @param {Integer} k
# @param {Integer[]} nums
# @return {Integer}
def magical_sum(m, k, nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer $m
* @param Integer $k
* @param Integer[] $nums
* @return Integer
*/
function magicalSum($m, $k, $nums) {

}
}
```

Dart:

```
class Solution {
int magicalSum(int m, int k, List<int> nums) {

}
}
```

Scala:

```
object Solution {
def magicalSum(m: Int, k: Int, nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec magical_sum(m :: integer, k :: integer, nums :: [integer]) :: integer
```

```
def magical_sum(m, k, nums) do

end
end
```

Erlang:

```
-spec magical_sum(M :: integer(), K :: integer(), Nums :: [integer()]) ->
integer().
magical_sum(M, K, Nums) ->
.
```

Racket:

```
(define/contract (magical-sum m k nums)
(-> exact-integer? exact-integer? (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Sum of Array Product of Magical Sequences
* Difficulty: Hard
* 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 magicalSum(int m, int k, vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Find Sum of Array Product of Magical Sequences
* Difficulty: Hard
* 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 magicalSum(int m, int k, int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Sum of Array Product of Magical Sequences
Difficulty: Hard
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 magicalSum(self, m: int, k: int, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Find Sum of Array Product of Magical Sequences
* Difficulty: Hard
* 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} m
* @param {number} k
* @param {number[]} nums
* @return {number}
*/
var magicalSum = function(m, k, nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Sum of Array Product of Magical Sequences
* Difficulty: Hard
* 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 magicalSum(m: number, k: number, nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Find Sum of Array Product of Magical Sequences
* Difficulty: Hard
* 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 MagicalSum(int m, int k, int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Find Sum of Array Product of Magical Sequences
* Difficulty: Hard
* 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 magicalSum(int m, int k, int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Find Sum of Array Product of Magical Sequences
// Difficulty: Hard
// 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 magicalSum(m int, k int, nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun magicalSum(m: Int, k: Int, nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func magicalSum(_ m: Int, _ k: Int, _ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find Sum of Array Product of Magical Sequences
// Difficulty: Hard
// 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 magical_sum(m: i32, k: i32, nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} m
# @param {Integer} k
# @param {Integer[]} nums
# @return {Integer}
def magical_sum(m, k, nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $m
* @param Integer $k
* @param Integer[] $nums
* @return Integer
*/
function magicalSum($m, $k, $nums) {

}
}
```

Dart Solution:

```
class Solution {
int magicalSum(int m, int k, List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def magicalSum(m: Int, k: Int, nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec magical_sum(m :: integer, k :: integer, nums :: [integer]) :: integer
def magical_sum(m, k, nums) do
```

```


end
end
```

Erlang Solution:

```
-spec magical_sum(M :: integer(), K :: integer(), Nums :: [integer()]) ->
integer().
magical_sum(M, K, Nums) ->
.
```

Racket Solution:

```
(define/contract (magical-sum m k nums)
(-> exact-integer? exact-integer? (listof exact-integer?) exact-integer?)
)
```