Problem 3082: Find the Sum of the Power of All Subsequences

Problem Information

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

Problem Description

You are given an integer array

nums

of length

n

and a

positive

integer

k

.

The

power

of an array of integers is defined as the number of

subsequences

with their sum

equal

to

k

.

Return

the

sum

of

power

of all subsequences of

nums

.

Since the answer may be very large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

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

Output:

6

Explanation:

There are

5

subsequences of nums with non-zero power:

The subsequence

[

1

,

2

,

3

]

has

2

subsequences with

sum == 3

:

[1,2,

3

]

and

[

1

,

2

,3]

.

The subsequence

[

1

,2,

3

]

has

1

subsequence with

sum == 3

:

[1,2,

3

]

.

The subsequence

[1,

2

,

3

]

has

1

subsequence with

sum == 3

:

[1,2,

3

]

.

The subsequence

[

1

,

2

,3]

has

1

subsequence with

sum == 3

:

[

1

,

2

,3]

.

The subsequence

[1,2,

3

]

has

1

subsequence with

sum == 3

:

[1,2,

3

]

.

Hence the answer is

2 + 1 + 1 + 1 + 1 = 6

.

Example 2:

Input:

nums = [2,3,3], k = 5

Output:

4

Explanation:

There are

3

subsequences of nums with non-zero power:

The subsequence

[

2

,

3

,

3

]

has 2 subsequences with

sum == 5

:

[

2

,3,

3

]

and

[

2

,

3

,3]

.

The subsequence

[

2

,3,

3

]

has 1 subsequence with

sum == 5

:

[

2

,3,

3

]

.

The subsequence

[

2

,

3

,3]

has 1 subsequence with

sum == 5

:

[

2

,

3

,3]

.

Hence the answer is

2 + 1 + 1 = 4

.

Example 3:

Input:

nums = [1,2,3], k = 7

Output:

0

Explanation:

There exists no subsequence with sum

7

. Hence all subsequences of nums have

power = 0

.

Constraints:

1 <= n <= 100

1 <= nums[i] <= 10

4

1 <= k <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
var sumOfPower = function(nums, k) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int sumOfPower(int* nums, int numsSize, int k) {

}
```

Go:

```
func sumOfPower(nums []int, k int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

```
}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (sum-of-power nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Sum of the Power of All Subsequences
* Difficulty: Hard
* Tags: array, dp
*
* 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 sumOfPower(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Sum of the Power of All Subsequences
* Difficulty: Hard
* Tags: array, dp
*
* 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 sumOfPower(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Sum of the Power of All Subsequences
Difficulty: Hard
Tags: array, dp

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 sumOfPower(self, nums: List[int], k: int) -> int:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the Sum of the Power of All Subsequences
* Difficulty: Hard
* Tags: array, dp
*
* 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} k
* @return {number}
*/
var sumOfPower = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Sum of the Power of All Subsequences
* Difficulty: Hard
* Tags: array, dp
*
```

```
* 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 sumOfPower(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Find the Sum of the Power of All Subsequences
* Difficulty: Hard
* Tags: array, dp
*
* 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 SumOfPower(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Find the Sum of the Power of All Subsequences
* Difficulty: Hard
* Tags: array, dp
*
* 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 sumOfPower(int* nums, int numsSize, int k) {
```

```
}
```

Go Solution:

```
// Problem: Find the Sum of the Power of All Subsequences
// Difficulty: Hard
// Tags: array, dp
//
// 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 sumOfPower(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find the Sum of the Power of All Subsequences
// Difficulty: Hard
// Tags: array, dp
//
// 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 sum_of_power(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (sum-of-power nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```