Problem 3351: Sum of Good Subsequences

Problem Information

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

Problem Description

You are given an integer array

nums

. A

good

subsequence

is defined as a subsequence of

nums

where the absolute difference between any

two

consecutive elements in the subsequence is

exactly

1.

Return the

sum

of all

possible

good subsequences

of

nums

.

Since the answer may be very large, return it

modulo

10

9

+ 7

.

Note

that a subsequence of size 1 is considered good by definition.

Example 1:

Input:

nums = [1,2,1]

Output:

14

Explanation:

Good subsequences are:

[1]

,

[2]

,

[1]

,

[1,2]

,

[2,1]

,

[1,2,1]

.

The sum of elements in these subsequences is 14.

Example 2:

Input:

nums = [3,4,5]

Output:

40

Explanation:

Good subsequences are:

[3]

,

[4]

,

[5]

,

[3,4]

,

[4,5]

,

[3,4,5]

.

The sum of elements in these subsequences is 40.

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
var sumOfGoodSubsequences = function(nums) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

```
}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (sum-of-good-subsequences nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of Good Subsequences
* Difficulty: Hard
* Tags: array, dp, 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 sumOfGoodSubsequences(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Good Subsequences
* Difficulty: Hard
* Tags: array, dp, 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 sumOfGoodSubsequences(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of Good Subsequences
Difficulty: Hard
Tags: array, dp, 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 sumOfGoodSubsequences(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Good Subsequences
* Difficulty: Hard
* Tags: array, dp, 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 sumOfGoodSubsequences = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Good Subsequences
* Difficulty: Hard
* Tags: array, dp, 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 sumOfGoodSubsequences(nums: number[]): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Sum of Good Subsequences
* Difficulty: Hard
* Tags: array, dp, 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 SumOfGoodSubsequences(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Sum of Good Subsequences
* Difficulty: Hard
* Tags: array, dp, 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 sumOfGoodSubsequences(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Sum of Good Subsequences
// Difficulty: Hard
```

```
// Tags: array, dp, 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 sumOfGoodSubsequences(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sum of Good Subsequences
// Difficulty: Hard
// Tags: array, dp, 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 sum_of_good_subsequences(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (sum-of-good-subsequences nums)
(-> (listof exact-integer?) exact-integer?)
)
```