Problem 3299: Sum of Consecutive Subsequences

Problem Information

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

Problem Description

We call an array

arr

of length

n

consecutive

if one of the following holds:

arr[i] - arr[i - 1] == 1

for

all

1 <= i < n

.

arr[i] - arr[i - 1] == -1

for

all

1 <= i < n

.

The

value

of an array is the sum of its elements.

For example,

[3, 4, 5]

is a consecutive array of value 12 and

[9, 8]

is another of value 17. While

[3, 4, 3]

and

[8, 6]

are not consecutive.

Given an array of integers

nums

, return the

sum

of the

values

of all

consecutive

non-empty

subsequences

.

Since the answer may be very large, return it

modulo

10

9

+ 7.

Note

that an array of length 1 is also considered consecutive.

Example 1:

Input:

nums = [1,2]

Output:

6

Explanation:

The consecutive subsequences are:

[1]

,

[2]

,

[1, 2]

.

Example 2:

Input:

nums = [1,4,2,3]

Output:

31

Explanation:

The consecutive subsequences are:

[1]

,

[4]

,

[2]

,

[3]

,

[1, 2]

,

[2, 3]

,

[4, 3]

,

[1, 2, 3]

.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
```

```
int getSum(vector<int>& nums) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```php
class Solution {

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

}
}
```

Dart:

```dart
class Solution {
int getSum(List<int> nums) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (get-sum nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of Consecutive 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 getSum(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Consecutive 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 getSum(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of Consecutive 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 getSum(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Consecutive 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 getSum = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Consecutive 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 getSum(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Sum of Consecutive 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 GetSum(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Sum of Consecutive 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 getSum(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Sum of Consecutive 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 getSum(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sum of Consecutive 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 get_sum(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (get-sum nums)
(-> (listof exact-integer?) exact-integer?)
)
```