Problem 3201: Find the Maximum Length of Valid Subsequence I

Problem Information

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

Problem Description

You are given an integer array

nums

.

A

subsequence

sub

of

nums

with length

x

is called

valid

if it satisfies:

(sub[0] + sub[1]) % 2 == (sub[1] + sub[2]) % 2 == ... == (sub[x - 2] + sub[x - 1]) % 2.

Return the length of the

longest

valid

subsequence of

nums

.

A

subsequence

is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.

Example 1:

Input:

nums = [1,2,3,4]

Output:

4

Explanation:

The longest valid subsequence is

[1, 2, 3, 4]

.

Example 2:

Input:

nums = [1,2,1,1,2,1,2]

Output:

6

Explanation:

The longest valid subsequence is

[1, 2, 1, 2, 1, 2]

.

Example 3:

Input:

nums = [1,3]

Output:

2

Explanation:

The longest valid subsequence is

[1, 3]

.

Constraints:

2 <= nums.length <= 2 * 10

5

1 <= nums[i] <= 10

7

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (maximum-length nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Maximum Length of Valid Subsequence I
* Difficulty: Medium
* 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 maximumLength(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Maximum Length of Valid Subsequence I
* Difficulty: Medium
* 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 maximumLength(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Maximum Length of Valid Subsequence I
Difficulty: Medium
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 maximumLength(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the Maximum Length of Valid Subsequence I
* Difficulty: Medium
* 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
* @return {number}
*/
var maximumLength = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Maximum Length of Valid Subsequence I
* Difficulty: Medium
* 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 maximumLength(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Find the Maximum Length of Valid Subsequence I
* Difficulty: Medium
* 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 MaximumLength(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Find the Maximum Length of Valid Subsequence I
* Difficulty: Medium
* 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 maximumLength(int* nums, int numsSize) {
}
```

Go Solution:

```
// Problem: Find the Maximum Length of Valid Subsequence I
// Difficulty: Medium
// 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 maximumLength(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find the Maximum Length of Valid Subsequence I
// Difficulty: Medium
// 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 maximum_length(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (maximum-length nums)
(-> (listof exact-integer?) exact-integer?)
)
```