Problem 3165: Maximum Sum of Subsequence With Non-adjacent Elements

Problem Information

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

Problem Description

You are given an array

nums

consisting of integers. You are also given a 2D array

queries

, where

queries[i] = [pos

i

, x

i

]

.

For query

i

, we first set

nums[pos

i

]

equal to

x

i

, then we calculate the answer to query

i

which is the

maximum

sum of a

subsequence

of

nums

where

no two adjacent elements are selected

.

Return the

sum

of the answers to all queries.

Since the final answer may be very large, return it

modulo

10

9

+ 7

.

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 = [3,5,9], queries = [[1,-2],[0,-3]]

Output:

21

Explanation:

After the 1

st

query,

nums = [3,-2,9]

and the maximum sum of a subsequence with non-adjacent elements is

3 + 9 = 12

.

After the 2

nd

query,

nums = [-3,-2,9]

and the maximum sum of a subsequence with non-adjacent elements is 9.

Example 2:

Input:

nums = [0,-1], queries = [[0,-5]]

Output:

0

Explanation:

After the 1

st

query,

nums = [-5,-1]

and the maximum sum of a subsequence with non-adjacent elements is 0 (choosing an empty subsequence).

Constraints:

1 <= nums.length <= 5 * 10

4

-10

5

<= nums[i] <= 10

5

1 <= queries.length <= 5 * 10

4

queries[i] == [pos

i

, x

i

]

0 <= pos

i

<= nums.length - 1

-10

5

<= x

i

<= 10

5

Code Snippets

C++:

```
class Solution {
public:
int maximumSumSubsequence(vector<int>& nums, vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public int maximumSumSubsequence(int[] nums, int[][] queries) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def maximumSumSubsequence(self, nums, queries):
"""
```

```
:type nums: List[int]
:type queries: List[List[int]]
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function maximumSumSubsequence(nums: number[], queries: number[][]): number {

};
```

C#:

```
public class Solution {
public int MaximumSumSubsequence(int[] nums, int[][] queries) {

}
}
```

C:

```
int maximumSumSubsequence(int* nums, int numsSize, int** queries, int
queriesSize, int* queriesColSize) {

}
```

Go:

```
func maximumSumSubsequence(nums []int, queries [][]int) int {

}
```

Kotlin:

```
class Solution {
fun maximumSumSubsequence(nums: IntArray, queries: Array<IntArray>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[][]} queries
# @return {Integer}
def maximum_sum_subsequence(nums, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[][] $queries
* @return Integer
*/
```

```
function maximumSumSubsequence($nums, $queries) {

}
}
```

Dart:

```
class Solution {
int maximumSumSubsequence(List<int> nums, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def maximumSumSubsequence(nums: Array[Int], queries: Array[Array[Int]]): Int
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_sum_subsequence(nums :: [integer], queries :: [[integer]]) ::
integer
def maximum_sum_subsequence(nums, queries) do

end
end
```

Erlang:

```
-spec maximum_sum_subsequence(Nums :: [integer()], Queries :: [[integer()]])
-> integer().
maximum_sum_subsequence(Nums, Queries) ->
.
```

Racket:

```
(define/contract (maximum-sum-subsequence nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Sum of Subsequence With Non-adjacent Elements
* Difficulty: Hard
* Tags: array, tree, 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 maximumSumSubsequence(vector<int>& nums, vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Sum of Subsequence With Non-adjacent Elements
* Difficulty: Hard
* Tags: array, tree, 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 maximumSumSubsequence(int[] nums, int[][] queries) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Maximum Sum of Subsequence With Non-adjacent Elements
Difficulty: Hard
Tags: array, tree, 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 maximumSumSubsequence(self, nums: List[int], queries: List[List[int]]) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Sum of Subsequence With Non-adjacent Elements
* Difficulty: Hard
* Tags: array, tree, 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[][]} queries
* @return {number}
*/
var maximumSumSubsequence = function(nums, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Sum of Subsequence With Non-adjacent Elements
* Difficulty: Hard
* Tags: array, tree, 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 maximumSumSubsequence(nums: number[], queries: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Sum of Subsequence With Non-adjacent Elements
* Difficulty: Hard
* Tags: array, tree, 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 MaximumSumSubsequence(int[] nums, int[][] queries) {
```

```
}
}
```

C Solution:

```
/*
* Problem: Maximum Sum of Subsequence With Non-adjacent Elements
* Difficulty: Hard
* Tags: array, tree, 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 maximumSumSubsequence(int* nums, int numsSize, int** queries, int
queriesSize, int* queriesColSize) {

}
```

Go Solution:

```
// Problem: Maximum Sum of Subsequence With Non-adjacent Elements
// Difficulty: Hard
// Tags: array, tree, 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 maximumSumSubsequence(nums []int, queries [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumSumSubsequence(nums: IntArray, queries: Array<IntArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Sum of Subsequence With Non-adjacent Elements
// Difficulty: Hard
// Tags: array, tree, 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_sum_subsequence(nums: Vec<i32>, queries: Vec<Vec<i32>>) -> i32
{

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[][]} queries
# @return {Integer}
def maximum_sum_subsequence(nums, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[][] $queries
* @return Integer
```

```
*/
function maximumSumSubsequence($nums, $queries) {

}
}
```

Dart Solution:

```
class Solution {
int maximumSumSubsequence(List<int> nums, List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def maximumSumSubsequence(nums: Array[Int], queries: Array[Array[Int]]): Int
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_sum_subsequence(nums :: [integer], queries :: [[integer]]) ::
integer
def maximum_sum_subsequence(nums, queries) do

end
end
```

Erlang Solution:

```
-spec maximum_sum_subsequence(Nums :: [integer()], Queries :: [[integer()]])
-> integer().
maximum_sum_subsequence(Nums, Queries) ->
.
```

Racket Solution:

```
(define/contract (maximum-sum-subsequence nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?)
)
```