Problem 1695: Maximum Erasure Value

Problem Information

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

Problem Description

You are given an array of positive integers

nums

and want to erase a subarray containing

unique elements

. The

score

you get by erasing the subarray is equal to the

sum

of its elements.

Return

the

maximum score

you can get by erasing

exactly one

subarray.

An array

b

is called to be a

subarray

of

a

if it forms a contiguous subsequence of

a

, that is, if it is equal to

a[l],a[l+1],...,a[r]

for some

(l,r)

.

Example 1:

Input:

nums = [4,2,4,5,6]

Output:

17

Explanation:

The optimal subarray here is [2,4,5,6].

Example 2:

Input:

nums = [5,2,1,2,5,2,1,2,5]

Output:

8

Explanation:

The optimal subarray here is [5,2,1] or [1,2,5].

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

4

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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


}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Erasure Value
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Maximum Erasure Value
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int maximumUniqueSubarray(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Erasure Value
Difficulty: Medium
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def maximumUniqueSubarray(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Erasure Value
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Erasure Value
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function maximumUniqueSubarray(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Erasure Value
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

public class Solution {
public int MaximumUniqueSubarray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Erasure Value
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int maximumUniqueSubarray(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximum Erasure Value
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func maximumUniqueSubarray(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Erasure Value
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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