Problem 3487: Maximum Unique Subarray Sum After Deletion

Problem Information

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

Problem Description

You are given an integer array

nums

.

You are allowed to delete any number of elements from

nums

without making it

empty

. After performing the deletions, select a

subarray

of

nums

such that:

All elements in the subarray are

unique

.

The sum of the elements in the subarray is

maximized

.

Return the

maximum sum

of such a subarray.

Example 1:

Input:

nums = [1,2,3,4,5]

Output:

15

Explanation:

Select the entire array without deleting any element to obtain the maximum sum.

Example 2:

Input:

nums = [1,1,0,1,1]

Output:

1

Explanation:

Delete the element

nums[0] == 1

,

nums[1] == 1

,

nums[2] == 0

, and

nums[3] == 1

. Select the entire array

[1]

to obtain the maximum sum.

Example 3:

Input:

nums = [1,2,-1,-2,1,0,-1]

Output:

3

Explanation:

Delete the elements

nums[2] == -1

and

nums[3] == -2

, and select the subarray

[2, 1]

from

[1, 2, 1, 0, -1]

to obtain the maximum sum.

Constraints:

1 <= nums.length <= 100

-100 <= nums[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Unique Subarray Sum After Deletion
* Difficulty: Easy
* Tags: array, greedy, 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 maxSum(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Unique Subarray Sum After Deletion
* Difficulty: Easy
* Tags: array, greedy, 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 maxSum(int[] nums) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Maximum Unique Subarray Sum After Deletion
Difficulty: Easy
Tags: array, greedy, 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 maxSum(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Unique Subarray Sum After Deletion
* Difficulty: Easy
* Tags: array, greedy, 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 maxSum = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Unique Subarray Sum After Deletion
* Difficulty: Easy
* Tags: array, greedy, 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 maxSum(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Unique Subarray Sum After Deletion
* Difficulty: Easy
* Tags: array, greedy, 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 MaxSum(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Unique Subarray Sum After Deletion
* Difficulty: Easy
* Tags: array, greedy, 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 maxSum(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximum Unique Subarray Sum After Deletion
// Difficulty: Easy
// Tags: array, greedy, 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 maxSum(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Maximum Unique Subarray Sum After Deletion
// Difficulty: Easy
// Tags: array, greedy, 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 max_sum(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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