Problem 3410: Maximize Subarray Sum After Removing All Occurrences of One Element

Problem Information

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

Problem Description

You are given an integer array

nums

.

You can do the following operation on the array

at most

once:

Choose

any

integer

x

such that

nums

remains

non-empty

on removing all occurrences of

x

.

Remove

all

occurrences of

x

from the array.

Return the

maximum

subarray

sum across

all

possible resulting arrays.

Example 1:

Input:

nums = [-3,2,-2,-1,3,-2,3]

Output:

7

Explanation:

We can have the following arrays after at most one operation:

The original array is

nums = [

-3, 2, -2, -1,

3, -2, 3

]

. The maximum subarray sum is

3 + (-2) + 3 = 4

.

Deleting all occurences of

x = -3

results in

nums = [2, -2, -1,

3, -2, 3

]

. The maximum subarray sum is

3 + (-2) + 3 = 4

.

Deleting all occurences of

x = -2

results in

nums = [

-3,

2, -1, 3, 3

]

. The maximum subarray sum is

2 + (-1) + 3 + 3 = 7

.

Deleting all occurences of

x = -1

results in

nums = [

-3, 2, -2,

3, -2, 3

]

. The maximum subarray sum is

3 + (-2) + 3 = 4

.

Deleting all occurences of

x = 3

results in

nums = [

-3,

2

, -2, -1, -2

]

. The maximum subarray sum is 2.

The output is

max(4, 4, 7, 4, 2) = 7

.

Example 2:

Input:

nums = [1,2,3,4]

Output:

10

Explanation:

It is optimal to not perform any operations.

Constraints:

1 <= nums.length <= 10

5

-10

6

<= nums[i] <= 10

6

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long maxSubarraySum(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long MaxSubarraySum(int[] nums) {

}
}
```

C:

```
long long maxSubarraySum(int* nums, int numsSize) {

}
```

Go:

```
func maxSubarraySum(nums []int) int64 {

}
```

Kotlin:

```
class Solution {
fun maxSubarraySum(nums: IntArray): Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn max_subarray_sum(nums: Vec<i32>) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximize Subarray Sum After Removing All Occurrences of One
Element
* 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:
long long maxSubarraySum(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize Subarray Sum After Removing All Occurrences of One
Element
* 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 long maxSubarraySum(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize Subarray Sum After Removing All Occurrences of One Element
```

```
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 maxSubarraySum(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximize Subarray Sum After Removing All Occurrences of One
Element
* 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
* @return {number}
*/
var maxSubarraySum = function(nums) {
};
```

TypeScript Solution:

```
/**
* Problem: Maximize Subarray Sum After Removing All Occurrences of One
Element
* 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 maxSubarraySum(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximize Subarray Sum After Removing All Occurrences of One
Element
* 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 long MaxSubarraySum(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximize Subarray Sum After Removing All Occurrences of One
Element
* 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
*/

long long maxSubarraySum(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximize Subarray Sum After Removing All Occurrences of One
Element
// 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 maxSubarraySum(nums []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maxSubarraySum(nums: IntArray): Long {

}
}
```

Swift Solution:

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



}
}
```

Rust Solution:

```
// Problem: Maximize Subarray Sum After Removing All Occurrences of One
Element
// 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 max_subarray_sum(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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