Problem 918: Maximum Sum Circular Subarray

Problem Information

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

Problem Description

Given a

circular integer array

nums

of length

n

, return

the maximum possible sum of a non-empty

subarray

of

nums

.

A

circular array

means the end of the array connects to the beginning of the array. Formally, the next element of

nums[i]

is

nums[(i + 1) % n]

and the previous element of

nums[i]

is

nums[(i - 1 + n) % n]

.

A

subarray

may only include each element of the fixed buffer

nums

at most once. Formally, for a subarray

nums[i], nums[i + 1], ..., nums[j]

, there does not exist

i <= k1

,

k2 <= j

with

k1 % n == k2 % n

.

Example 1:

Input:

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

Output:

3

Explanation:

Subarray [3] has maximum sum 3.

Example 2:

Input:

nums = [5,-3,5]

Output:

10

Explanation:

Subarray [5,5] has maximum sum 5 + 5 = 10.

Example 3:

Input:

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

Output:

-2

Explanation:

Subarray [-2] has maximum sum -2.

Constraints:

n == nums.length

1 <= n <= 3 * 10

4

-3 * 10

4

<= nums[i] <= 3 * 10

4

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int maxSubarraySumCircular(int[] nums) {
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Sum Circular Subarray
* Difficulty: Medium
* Tags: array, dp, queue
*
* 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 maxSubarraySumCircular(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Sum Circular Subarray
* Difficulty: Medium
* Tags: array, dp, queue
*
* 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 maxSubarraySumCircular(int[] nums) {
}
}
```

Python3 Solution:

```
"""
Problem: Maximum Sum Circular Subarray
Difficulty: Medium
Tags: array, dp, queue

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Sum Circular Subarray
* Difficulty: Medium
* Tags: array, dp, queue
*
* 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 maxSubarraySumCircular = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Sum Circular Subarray
* Difficulty: Medium
* Tags: array, dp, queue
*
* 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 maxSubarraySumCircular(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Sum Circular Subarray
* Difficulty: Medium
* Tags: array, dp, queue
*
* 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 MaxSubarraySumCircular(int[] nums) {

}

}
```

C Solution:

```
/*
* Problem: Maximum Sum Circular Subarray
* Difficulty: Medium
* Tags: array, dp, queue
*
* 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 maxSubarraySumCircular(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximum Sum Circular Subarray
// Difficulty: Medium
// Tags: array, dp, queue
//
// 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 maxSubarraySumCircular(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Sum Circular Subarray
// Difficulty: Medium
// Tags: array, dp, queue
//
// 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_circular(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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