Problem 2382: Maximum Segment Sum After Removals

Problem Information

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

Problem Description

You are given two

0-indexed

integer arrays

nums

and

removeQueries

, both of length

n

. For the

i

th

query, the element in

nums

at the index

removeQueries[i]

is removed, splitting

nums

into different segments.

A

segment

is a contiguous sequence of

positive

integers in

nums

. A

segment sum

is the sum of every element in a segment.

Return

an integer array

answer

, of length

n

, where

answer[i]

is the

maximum

segment sum after applying the

i

th

removal.

Note:

The same index will

not

be removed more than once.

Example 1:

Input:

nums = [1,2,5,6,1], removeQueries = [0,3,2,4,1]

Output:

[14,7,2,2,0]

Explanation:

Using 0 to indicate a removed element, the answer is as follows: Query 1: Remove the 0th element, nums becomes [0,2,5,6,1] and the maximum segment sum is 14 for segment [2,5,6,1]. Query 2: Remove the 3rd element, nums becomes [0,2,5,0,1] and the maximum

segment sum is 7 for segment [2,5]. Query 3: Remove the 2nd element, nums becomes [0,2,0,0,1] and the maximum segment sum is 2 for segment [2]. Query 4: Remove the 4th element, nums becomes [0,2,0,0,0] and the maximum segment sum is 2 for segment [2]. Query 5: Remove the 1st element, nums becomes [0,0,0,0,0] and the maximum segment sum is 0, since there are no segments. Finally, we return [14,7,2,2,0].

Example 2:

Input:

nums = [3,2,11,1], removeQueries = [3,2,1,0]

Output:

[16,5,3,0]

Explanation:

Using 0 to indicate a removed element, the answer is as follows: Query 1: Remove the 3rd element, nums becomes [3,2,11,0] and the maximum segment sum is 16 for segment [3,2,11]. Query 2: Remove the 2nd element, nums becomes [3,2,0,0] and the maximum segment sum is 5 for segment [3,2]. Query 3: Remove the 1st element, nums becomes [3,0,0,0] and the maximum segment sum is 3 for segment [3]. Query 4: Remove the 0th element, nums becomes [0,0,0,0] and the maximum segment sum is 0, since there are no segments. Finally, we return [16,5,3,0].

Constraints:

n == nums.length == removeQueries.length

1 <= n <= 10

5

1 <= nums[i] <= 10

9

0 <= removeQueries[i] < n

All the values of

removeQueries

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
vector<long long> maximumSegmentSum(vector<int>& nums, vector<int>&
removeQueries) {

}
};
```

Java:

```
class Solution {
public long[] maximumSegmentSum(int[] nums, int[] removeQueries) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def maximumSegmentSum(self, nums, removeQueries):
```

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

JavaScript:

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

};
```

TypeScript:

```
function maximumSegmentSum(nums: number[], removeQueries: number[]): number[]
{

};
```

C#:

```
public class Solution {
public long[] MaximumSegmentSum(int[] nums, int[] removeQueries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* maximumSegmentSum(int* nums, int numsSize, int* removeQueries, int
removeQueriesSize, int* returnSize) {

}
```

Go:

```
func maximumSegmentSum(nums []int, removeQueries []int) []int64 {

}
```

Kotlin:

```
class Solution {
fun maximumSegmentSum(nums: IntArray, removeQueries: IntArray): LongArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[]} remove_queries
# @return {Integer[]}
def maximum_segment_sum(nums, remove_queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $removeQueries
* @return Integer[]
*/
function maximumSegmentSum($nums, $removeQueries) {

}
}
```

Dart:

```
class Solution {
List<int> maximumSegmentSum(List<int> nums, List<int> removeQueries) {

}
}
```

Scala:

```
object Solution {
def maximumSegmentSum(nums: Array[Int], removeQueries: Array[Int]):
Array[Long] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_segment_sum(nums :: [integer], remove_queries :: [integer]) ::
[integer]
def maximum_segment_sum(nums, remove_queries) do

end
end
```

Erlang:

```
-spec maximum_segment_sum(Nums :: [integer()], RemoveQueries :: [integer()])
-> [integer()].
```

```
maximum_segment_sum(Nums, RemoveQueries) ->
.
```

Racket:

```
(define/contract (maximum-segment-sum nums removeQueries)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Segment Sum After Removals
* Difficulty: Hard
* Tags: array, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<long long> maximumSegmentSum(vector<int>& nums, vector<int>&
removeQueries) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Segment Sum After Removals
* Difficulty: Hard
* Tags: array, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public long[] maximumSegmentSum(int[] nums, int[] removeQueries) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Segment Sum After Removals
Difficulty: Hard
Tags: array, graph

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Segment Sum After Removals
```

```
* Difficulty: Hard
* Tags: array, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Segment Sum After Removals
* Difficulty: Hard
* Tags: array, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function maximumSegmentSum(nums: number[], removeQueries: number[]): number[]
{

};
```

C# Solution:

```
/*
* Problem: Maximum Segment Sum After Removals
* Difficulty: Hard
* Tags: array, graph
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public long[] MaximumSegmentSum(int[] nums, int[] removeQueries) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Segment Sum After Removals
* Difficulty: Hard
* Tags: array, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* maximumSegmentSum(int* nums, int numsSize, int* removeQueries, int
removeQueriesSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Maximum Segment Sum After Removals
// Difficulty: Hard
// Tags: array, graph
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach
```

```
func maximumSegmentSum(nums []int, removeQueries []int) []int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maximumSegmentSum(nums: IntArray, removeQueries: IntArray): LongArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Segment Sum After Removals
// Difficulty: Hard
// Tags: array, graph
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn maximum_segment_sum(nums: Vec<i32>, remove_queries: Vec<i32>) ->
Vec<i64> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[]} remove_queries
```

```
# @return {Integer[]}
def maximum_segment_sum(nums, remove_queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $removeQueries
* @return Integer[]
*/
function maximumSegmentSum($nums, $removeQueries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> maximumSegmentSum(List<int> nums, List<int> removeQueries) {

}
}
```

Scala Solution:

```
object Solution {
def maximumSegmentSum(nums: Array[Int], removeQueries: Array[Int]):
Array[Long] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_segment_sum(nums :: [integer], remove_queries :: [integer]) ::
[integer]
```

```
def maximum_segment_sum(nums, remove_queries) do

end
end
```

Erlang Solution:

```
-spec maximum_segment_sum(Nums :: [integer()], RemoveQueries :: [integer()])
-> [integer()].
maximum_segment_sum(Nums, RemoveQueries) ->
.
```

Racket Solution:

```
(define/contract (maximum-segment-sum nums removeQueries)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```