Problem 454: 4Sum II

Problem Information

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

Problem Description

Given four integer arrays

nums1

,

nums2

,

nums3

, and

nums4

all of length

n

, return the number of tuples

(i, j, k, l)

such that:

0 <= i, j, k, l < n

nums1[i] + nums2[j] + nums3[k] + nums4[l] == 0

Example 1:

Input:

nums1 = [1,2], nums2 = [-2,-1], nums3 = [-1,2], nums4 = [0,2]

Output:

2

Explanation:

The two tuples are: 1. (0, 0, 0, 1) -> nums1[0] + nums2[0] + nums3[0] + nums4[1] = 1 + (-2) + (-1) + 2 = 0 2. (1, 1, 0, 0) -> nums1[1] + nums2[1] + nums3[0] + nums4[0] = 2 + (-1) + (-1) + 0 = 0

Example 2:

Input:

nums1 = [0], nums2 = [0], nums3 = [0], nums4 = [0]

Output:

1

Constraints:

n == nums1.length

n == nums2.length

n == nums3.length

n == nums4.length

1 <= n <= 200

-2

28

<= nums1[i], nums2[i], nums3[i], nums4[i] <= 2

28

Code Snippets

C++:

```
class Solution {
public:
int fourSumCount(vector<int>& nums1, vector<int>& nums2, vector<int>& nums3,
vector<int>& nums4) {

}
};
```

Java:

```
class Solution {
public int fourSumCount(int[] nums1, int[] nums2, int[] nums3, int[] nums4) {

}
}
```

Python3:

```
class Solution:
def fourSumCount(self, nums1: List[int], nums2: List[int], nums3: List[int],
nums4: List[int]) -> int:
```

Python:

```
class Solution(object):
def fourSumCount(self, nums1, nums2, nums3, nums4):
```

```
"""
:type nums1: List[int]
:type nums2: List[int]
:type nums3: List[int]
:type nums4: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums1
* @param {number[]} nums2
* @param {number[]} nums3
* @param {number[]} nums4
* @return {number}
*/
var fourSumCount = function(nums1, nums2, nums3, nums4) {

};
```

TypeScript:

```
function fourSumCount(nums1: number[], nums2: number[], nums3: number[],
nums4: number[]): number {

};
```

C#:

```
public class Solution {
public int FourSumCount(int[] nums1, int[] nums2, int[] nums3, int[] nums4) {

}
}
```

C:

```
int fourSumCount(int* nums1, int nums1Size, int* nums2, int nums2Size, int*
nums3, int nums3Size, int* nums4, int nums4Size) {

}
```

Go:

```
func fourSumCount(nums1 []int, nums2 []int, nums3 []int, nums4 []int) int {

}
```

Kotlin:

```
class Solution {
fun fourSumCount(nums1: IntArray, nums2: IntArray, nums3: IntArray, nums4:
IntArray): Int {

}
}
```

Swift:

```
class Solution {
func fourSumCount(_ nums1: [Int], _ nums2: [Int], _ nums3: [Int], _ nums4:
[Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn four_sum_count(nums1: Vec<i32>, nums2: Vec<i32>, nums3: Vec<i32>,
nums4: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @param {Integer[]} nums3
# @param {Integer[]} nums4
# @return {Integer}
def four_sum_count(nums1, nums2, nums3, nums4)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @param Integer[] $nums3
* @param Integer[] $nums4
* @return Integer
*/
function fourSumCount($nums1, $nums2, $nums3, $nums4) {

}
}
```

Dart:

```
class Solution {
int fourSumCount(List<int> nums1, List<int> nums2, List<int> nums3, List<int>
nums4) {

}
}
```

Scala:

```
object Solution {
def fourSumCount(nums1: Array[Int], nums2: Array[Int], nums3: Array[Int],
nums4: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec four_sum_count(nums1 :: [integer], nums2 :: [integer], nums3 ::
[integer], nums4 :: [integer]) :: integer
def four_sum_count(nums1, nums2, nums3, nums4) do

end
end
```

Erlang:

```
-spec four_sum_count(Nums1 :: [integer()], Nums2 :: [integer()], Nums3 ::
[integer()], Nums4 :: [integer()]) -> integer().
four_sum_count(Nums1, Nums2, Nums3, Nums4) ->
.
```

Racket:

```
(define/contract (four-sum-count nums1 nums2 nums3 nums4)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?)
(listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: 4Sum II
* 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 fourSumCount(vector<int>& nums1, vector<int>& nums2, vector<int>& nums3,
vector<int>& nums4) {

}
};
```

Java Solution:

```
/**
* Problem: 4Sum II
* 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 fourSumCount(int[] nums1, int[] nums2, int[] nums3, int[] nums4) {

}
}
```

Python3 Solution:

```
"""
Problem: 4Sum II
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 fourSumCount(self, nums1: List[int], nums2: List[int], nums3: List[int],
nums4: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def fourSumCount(self, nums1, nums2, nums3, nums4):
"""
:type nums1: List[int]
:type nums2: List[int]
:type nums3: List[int]
:type nums4: List[int]
:rtype: int
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: 4Sum II
* 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[]} nums1
* @param {number[]} nums2
* @param {number[]} nums3
* @param {number[]} nums4
* @return {number}
*/
var fourSumCount = function(nums1, nums2, nums3, nums4) {

};
```

TypeScript Solution:

```
/**
* Problem: 4Sum II
* 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 fourSumCount(nums1: number[], nums2: number[], nums3: number[],
nums4: number[]): number {

};
```

C# Solution:

```
/*
* Problem: 4Sum II
* 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 FourSumCount(int[] nums1, int[] nums2, int[] nums3, int[] nums4) {

}
}
```

C Solution:

```
/*
* Problem: 4Sum II
* 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 fourSumCount(int* nums1, int nums1Size, int* nums2, int nums2Size, int*
nums3, int nums3Size, int* nums4, int nums4Size) {

}
```

Go Solution:

```
// Problem: 4Sum II
// 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 fourSumCount(nums1 []int, nums2 []int, nums3 []int, nums4 []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun fourSumCount(nums1: IntArray, nums2: IntArray, nums3: IntArray, nums4:
IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func fourSumCount(_ nums1: [Int], _ nums2: [Int], _ nums3: [Int], _ nums4:
[Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: 4Sum II
// 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 four_sum_count(nums1: Vec<i32>, nums2: Vec<i32>, nums3: Vec<i32>,
nums4: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @param {Integer[]} nums3
# @param {Integer[]} nums4
# @return {Integer}
def four_sum_count(nums1, nums2, nums3, nums4)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @param Integer[] $nums3
* @param Integer[] $nums4
* @return Integer
*/
function fourSumCount($nums1, $nums2, $nums3, $nums4) {

}
}
```

Dart Solution:

```
class Solution {
int fourSumCount(List<int> nums1, List<int> nums2, List<int> nums3, List<int>
nums4) {

}
}
```

Scala Solution:

```
object Solution {
def fourSumCount(nums1: Array[Int], nums2: Array[Int], nums3: Array[Int],
nums4: Array[Int]): Int = {
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec four_sum_count(nums1 :: [integer], nums2 :: [integer], nums3 ::
[integer], nums4 :: [integer]) :: integer
def four_sum_count(nums1, nums2, nums3, nums4) do

end
end
```

Erlang Solution:

```
-spec four_sum_count(Nums1 :: [integer()], Nums2 :: [integer()], Nums3 ::
[integer()], Nums4 :: [integer()]) -> integer().
four_sum_count(Nums1, Nums2, Nums3, Nums4) ->
.
```

Racket Solution:

```
(define/contract (four-sum-count nums1 nums2 nums3 nums4)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?)
(listof exact-integer?) exact-integer?)
)
```