Problem 923: 3Sum With Multiplicity

Problem Information

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

Problem Description

Given an integer array

arr

, and an integer

target

, return the number of tuples

i, j, k

such that

i < j < k

and

arr[i] + arr[j] + arr[k] == target

.

As the answer can be very large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

arr = [1,1,2,2,3,3,4,4,5,5], target = 8

Output:

20

Explanation:

Enumerating by the values (arr[i], arr[j], arr[k]): (1, 2, 5) occurs 8 times; (1, 3, 4) occurs 8 times; (2, 2, 4) occurs 2 times; (2, 3, 3) occurs 2 times.

Example 2:

Input:

arr = [1,1,2,2,2,2], target = 5

Output:

12

Explanation:

arr[i] = 1, arr[j] = arr[k] = 2 occurs 12 times: We choose one 1 from [1,1] in 2 ways, and two 2s from [2,2,2,2] in 6 ways.

Example 3:

Input:

arr = [2,1,3], target = 6

Output:

1

Explanation:

(1, 2, 3) occured one time in the array so we return 1.

Constraints:

3 <= arr.length <= 3000

0 <= arr[i] <= 100

0 <= target <= 300

Code Snippets

C++:

```
class Solution {
public:
int threeSumMulti(vector<int>& arr, int target) {

}
};
```

Java:

```
class Solution {
public int threeSumMulti(int[] arr, int target) {

}
}
```

Python3:

```
class Solution:
def threeSumMulti(self, arr: List[int], target: int) -> int:
```

Python:

```
class Solution(object):
def threeSumMulti(self, arr, target):
"""
:type arr: List[int]
:type target: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} arr
* @param {number} target
* @return {number}
*/
var threeSumMulti = function(arr, target) {

};
```

TypeScript:

```
function threeSumMulti(arr: number[], target: number): number {

};
```

C#:

```
public class Solution {
public int ThreeSumMulti(int[] arr, int target) {

}
}
```

C:

```
int threeSumMulti(int* arr, int arrSize, int target) {

}
```

Go:

```
func threeSumMulti(arr []int, target int) int {

}
```

Kotlin:

```
class Solution {
fun threeSumMulti(arr: IntArray, target: Int): Int {

}
}
```

Swift:

```
class Solution {
func threeSumMulti(_ arr: [Int], _ target: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn three_sum_multi(arr: Vec<i32>, target: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @param {Integer} target
# @return {Integer}
def three_sum_multi(arr, target)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $target
* @return Integer
*/
function threeSumMulti($arr, $target) {

}
}
```

Dart:

```
class Solution {
int threeSumMulti(List<int> arr, int target) {

}
}
```

Scala:

```
object Solution {
def threeSumMulti(arr: Array[Int], target: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec three_sum_multi(arr :: [integer], target :: integer) :: integer
def three_sum_multi(arr, target) do

end
end
```

Erlang:

```
-spec three_sum_multi(Arr :: [integer()], Target :: integer()) -> integer().
three_sum_multi(Arr, Target) ->
```

```
.
```

Racket:

```
(define/contract (three-sum-multi arr target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: 3Sum With Multiplicity
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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 threeSumMulti(vector<int>& arr, int target) {

}
};
```

Java Solution:

```
/**
* Problem: 3Sum With Multiplicity
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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 threeSumMulti(int[] arr, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: 3Sum With Multiplicity
Difficulty: Medium
Tags: array, hash, sort

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 threeSumMulti(self, arr: List[int], target: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def threeSumMulti(self, arr, target):
"""
:type arr: List[int]
:type target: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: 3Sum With Multiplicity
* Difficulty: Medium
* Tags: array, hash, sort
*
```

```
* 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[]} arr
* @param {number} target
* @return {number}
*/
var threeSumMulti = function(arr, target) {

};
```

TypeScript Solution:

```
/**
* Problem: 3Sum With Multiplicity
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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 threeSumMulti(arr: number[], target: number): number {

};
```

C# Solution:

```
/*
* Problem: 3Sum With Multiplicity
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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 ThreeSumMulti(int[] arr, int target) {

}
}
```

C Solution:

```
/*
* Problem: 3Sum With Multiplicity
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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 threeSumMulti(int* arr, int arrSize, int target) {

}
```

Go Solution:

```
// Problem: 3Sum With Multiplicity
// Difficulty: Medium
// Tags: array, hash, sort
//
// 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 threeSumMulti(arr []int, target int) int {

}
```

Kotlin Solution:

```
class Solution {
fun threeSumMulti(arr: IntArray, target: Int): Int {
}
}
```

Swift Solution:

```
class Solution {
func threeSumMulti(_ arr: [Int], _ target: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: 3Sum With Multiplicity
// Difficulty: Medium
// Tags: array, hash, sort
//
// 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 three_sum_multi(arr: Vec<i32>, target: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @param {Integer} target
# @return {Integer}
def three_sum_multi(arr, target)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $arr
* @param Integer $target
* @return Integer
*/
function threeSumMulti($arr, $target) {

}
}
```

Dart Solution:

```
class Solution {
int threeSumMulti(List<int> arr, int target) {

}
}
```

Scala Solution:

```
object Solution {
def threeSumMulti(arr: Array[Int], target: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec three_sum_multi(arr :: [integer], target :: integer) :: integer
def three_sum_multi(arr, target) do

end
end
```

Erlang Solution:

```
-spec three_sum_multi(Arr :: [integer()], Target :: integer()) -> integer().
three_sum_multi(Arr, Target) ->
.
```

Racket Solution:

```
(define/contract (three-sum-multi arr target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```