Problem 1711: Count Good Meals

Problem Information

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

Problem Description

A

good meal

is a meal that contains

exactly two different food items

with a sum of deliciousness equal to a power of two.

You can pick

any

two different foods to make a good meal.

Given an array of integers

deliciousness

where

deliciousness[i]

is the deliciousness of the

i

th

item of food, return

the number of different

good meals

you can make from this list modulo

10

9

+ 7

.

Note that items with different indices are considered different even if they have the same deliciousness value.

Example 1:

Input:

deliciousness = [1,3,5,7,9]

Output:

4

Explanation:

The good meals are (1,3), (1,7), (3,5) and, (7,9). Their respective sums are 4, 8, 8, and 16, all of which are powers of 2.

Example 2:

Input:

deliciousness = [1,1,1,3,3,3,7]

Output:

15

Explanation:

The good meals are (1,1) with 3 ways, (1,3) with 9 ways, and (1,7) with 3 ways.

Constraints:

1 <= deliciousness.length <= 10

5

0 <= deliciousness[i] <= 2

20

Code Snippets

C++:

```
class Solution {
public:
int countPairs(vector<int>& deliciousness) {

}
};
```

Java:

```
class Solution {
public int countPairs(int[] deliciousness) {
```

```
}
}
```

Python3:

```
class Solution:
def countPairs(self, deliciousness: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function countPairs(deliciousness: number[]): number {

};
```

C#:

```
public class Solution {
public int CountPairs(int[] deliciousness) {

}
}
```

C:

```
int countPairs(int* deliciousness, int deliciousnessSize) {

}
```

Go:

```
func countPairs(deliciousness []int) int {

}
```

Kotlin:

```
class Solution {
fun countPairs(deliciousness: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func countPairs(_ deliciousness: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_pairs(deliciousness: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} deliciousness
# @return {Integer}
def count_pairs(deliciousness)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $deliciousness
* @return Integer
*/
function countPairs($deliciousness) {

}
}
```

Dart:

```
class Solution {
int countPairs(List<int> deliciousness) {

}
}
```

Scala:

```
object Solution {
def countPairs(deliciousness: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_pairs(deliciousness :: [integer]) :: integer
def count_pairs(deliciousness) do

end
end
```

Erlang:

```
-spec count_pairs(Deliciousness :: [integer()]) -> integer().
count_pairs(Deliciousness) ->
.
```

Racket:

```
(define/contract (count-pairs deliciousness)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Good Meals
* 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 countPairs(vector<int>& deliciousness) {

}
};
```

Java Solution:

```
/**
* Problem: Count Good Meals
* 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 countPairs(int[] deliciousness) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Count Good Meals
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 countPairs(self, deliciousness: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Good Meals
* 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[]} deliciousness
* @return {number}
*/
var countPairs = function(deliciousness) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Good Meals
* 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 countPairs(deliciousness: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Good Meals
* 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 CountPairs(int[] deliciousness) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Count Good Meals
* 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 countPairs(int* deliciousness, int deliciousnessSize) {

}
```

Go Solution:

```
// Problem: Count Good Meals
// 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 countPairs(deliciousness []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countPairs(deliciousness: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func countPairs(_ deliciousness: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Good Meals
// 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 count_pairs(deliciousness: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} deliciousness
# @return {Integer}
def count_pairs(deliciousness)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $deliciousness
* @return Integer
*/
function countPairs($deliciousness) {

}
}
```

Dart Solution:

```
class Solution {
int countPairs(List<int> deliciousness) {

}
}
```

Scala Solution:

```
object Solution {
def countPairs(deliciousness: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_pairs(deliciousness :: [integer]) :: integer
def count_pairs(deliciousness) do

end
end
```

Erlang Solution:

```
-spec count_pairs(Deliciousness :: [integer()]) -> integer().
count_pairs(Deliciousness) ->
.
```

Racket Solution:

```
(define/contract (count-pairs deliciousness)
(-> (listof exact-integer?) exact-integer?)
)
```