Problem 2748: Number of Beautiful Pairs

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

. A pair of indices

i

,

j

where

0 <= i < j < nums.length

is called beautiful if the

first digit

of

nums[i]

and the

last digit

of

nums[j]

are

coprime

.

Return

the total number of beautiful pairs in

nums

.

Two integers

x

and

y

are

coprime

if there is no integer greater than 1 that divides both of them. In other words,

x

and

y

are coprime if

gcd(x, y) == 1

, where

gcd(x, y)

is the

greatest common divisor

of

x

and

y

.

Example 1:

Input:

nums = [2,5,1,4]

Output:

5

Explanation:

There are 5 beautiful pairs in nums: When i = 0 and j = 1: the first digit of nums[0] is 2, and the last digit of nums[1] is 5. We can confirm that 2 and 5 are coprime, since gcd(2,5) == 1. When i = 0 and j = 2: the first digit of nums[0] is 2, and the last digit of nums[2] is 1. Indeed, gcd(2,1) == 1. When i = 1 and j = 2: the first digit of nums[1] is 5, and the last digit of nums[2] is 1. Indeed, gcd(5,1) == 1. When i = 1 and j = 3: the first digit of nums[1] is 5, and the last digit of nums[3] is 4. Indeed, gcd(5,4) == 1. When i = 2 and j = 3: the first digit of nums[2] is 1, and the last digit of nums[3] is 4. Indeed, gcd(1,4) == 1. Thus, we return 5.

Example 2:

Input:

nums = [11,21,12]

Output:

2

Explanation:

There are 2 beautiful pairs: When i = 0 and j = 1: the first digit of nums[0] is 1, and the last digit of nums[1] is 1. Indeed, gcd(1,1) == 1. When i = 0 and j = 2: the first digit of nums[0] is 1, and the last digit of nums[2] is 2. Indeed, gcd(1,2) == 1. Thus, we return 2.

Constraints:

2 <= nums.length <= 100

1 <= nums[i] <= 9999

nums[i] % 10 != 0

Code Snippets

C++:

```
class Solution {
public:
int countBeautifulPairs(vector<int>& nums) {
}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Number of Beautiful Pairs
* Difficulty: Easy
* Tags: array, math, 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 countBeautifulPairs(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Beautiful Pairs
* Difficulty: Easy
* Tags: array, math, 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 countBeautifulPairs(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Beautiful Pairs
Difficulty: Easy
Tags: array, math, 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 countBeautifulPairs(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Beautiful Pairs
* Difficulty: Easy
* Tags: array, math, 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[]} nums
* @return {number}
*/
var countBeautifulPairs = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Beautiful Pairs
* Difficulty: Easy
* Tags: array, math, 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 countBeautifulPairs(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Beautiful Pairs
* Difficulty: Easy
* Tags: array, math, 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 CountBeautifulPairs(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Number of Beautiful Pairs
* Difficulty: Easy
* Tags: array, math, 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 countBeautifulPairs(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Number of Beautiful Pairs
// Difficulty: Easy
// Tags: array, math, 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 countBeautifulPairs(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Beautiful Pairs
// Difficulty: Easy
// Tags: array, math, 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_beautiful_pairs(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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