Problem 1128: Number of Equivalent Domino Pairs

Problem Information

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

Problem Description

Given a list of

dominoes

,

dominoes[i] = [a, b]

is

equivalent to

dominoes[j] = [c, d]

if and only if either (

a == c

and

b == d

), or (

a == d

and

b == c

) - that is, one domino can be rotated to be equal to another domino.

Return

the number of pairs

(i, j)

for which

0 <= i < j < dominoes.length

, and

dominoes[i]

is

equivalent to

dominoes[j]

.

Example 1:

Input:

dominoes = [[1,2],[2,1],[3,4],[5,6]]

Output:

1

Example 2:

Input:

dominoes = [[1,2],[1,2],[1,1],[1,2],[2,2]]

Output:

3

Constraints:

1 <= dominoes.length <= 4 * 10

4

dominoes[i].length == 2

1 <= dominoes[i][j] <= 9

Code Snippets

C++:

```
class Solution {
public:
int numEquivDominoPairs(vector<vector<int>>& dominoes) {

}
};
```

Java:

```
class Solution {
public int numEquivDominoPairs(int[][] dominoes) {

}
}
```

Python3:

```
class Solution:
def numEquivDominoPairs(self, dominoes: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function numEquivDominoPairs(dominoes: number[][]): number {

};
```

C#:

```
public class Solution {
public int NumEquivDominoPairs(int[][] dominoes) {

}
}
```

C:

```
int numEquivDominoPairs(int** dominoes, int dominoesSize, int*
dominoesColSize) {
```

```
}
```

Go:

```
func numEquivDominoPairs(dominoes [][]int) int {

}
```

Kotlin:

```
class Solution {
fun numEquivDominoPairs(dominoes: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func numEquivDominoPairs(_ dominoes: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_equiv_domino_pairs(dominoes: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} dominoes
# @return {Integer}
def num_equiv_domino_pairs(dominoes)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $dominoes
* @return Integer
*/
function numEquivDominoPairs($dominoes) {

}
}
```

Dart:

```
class Solution {
int numEquivDominoPairs(List<List<int>> dominoes) {

}
}
```

Scala:

```
object Solution {
def numEquivDominoPairs(dominoes: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_equiv_domino_pairs(dominoes :: [[integer]]) :: integer
def num_equiv_domino_pairs(dominoes) do

end
end
```

Erlang:

```
-spec num_equiv_domino_pairs(Dominoes :: [[integer()]]) -> integer().
num_equiv_domino_pairs(Dominoes) ->
.
```

Racket:

```
(define/contract (num-equiv-domino-pairs dominoes)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Equivalent Domino Pairs
* Difficulty: Easy
* 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 numEquivDominoPairs(vector<vector<int>>& dominoes) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Equivalent Domino Pairs
* Difficulty: Easy
* 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 numEquivDominoPairs(int[][] dominoes) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Number of Equivalent Domino Pairs
Difficulty: Easy
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 numEquivDominoPairs(self, dominoes: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Equivalent Domino Pairs
* Difficulty: Easy
* 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[][]} dominoes
* @return {number}
*/
var numEquivDominoPairs = function(dominoes) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Equivalent Domino Pairs
* Difficulty: Easy
* 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 numEquivDominoPairs(dominoes: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Equivalent Domino Pairs
* Difficulty: Easy
* 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 NumEquivDominoPairs(int[][] dominoes) {

}
}
```

C Solution:

```
/*
* Problem: Number of Equivalent Domino Pairs
* Difficulty: Easy
* 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 numEquivDominoPairs(int** dominoes, int dominoesSize, int*
dominoesColSize) {

}
```

Go Solution:

```
// Problem: Number of Equivalent Domino Pairs
// Difficulty: Easy
// 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 numEquivDominoPairs(dominoes [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numEquivDominoPairs(dominoes: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func numEquivDominoPairs(_ dominoes: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Equivalent Domino Pairs
// Difficulty: Easy
// 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 num_equiv_domino_pairs(dominoes: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} dominoes
# @return {Integer}
def num_equiv_domino_pairs(dominoes)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $dominoes
* @return Integer
*/
function numEquivDominoPairs($dominoes) {

}
}
```

Dart Solution:

```
class Solution {
int numEquivDominoPairs(List<List<int>> dominoes) {

}
}
```

Scala Solution:

```
object Solution {
def numEquivDominoPairs(dominoes: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_equiv_domino_pairs(dominoes :: [[integer]]) :: integer
def num_equiv_domino_pairs(dominoes) do

end
end
```

Erlang Solution:

```
-spec num_equiv_domino_pairs(Dominoes :: [[integer()]]) -> integer().
num_equiv_domino_pairs(Dominoes) ->
.
```

Racket Solution:

```
(define/contract (num-equiv-domino-pairs dominoes)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```