Problem 2857: Count Pairs of Points With Distance k

Problem Information

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

Problem Description

You are given a

2D

integer array

coordinates

and an integer

k

, where

coordinates[i] = [x

i

, y

i

]

are the coordinates of the

i

th

point in a 2D plane.

We define the

distance

between two points

(x

1

, y

1

)

and

(x

2

, y

2

)

as

(x1 XOR x2) + (y1 XOR y2)

where

XOR

is the bitwise

XOR

operation.

Return

the number of pairs

(i, j)

such that

i < j

and the distance between points

i

and

j

is equal to

k

.

Example 1:

Input:

coordinates = [[1,2],[4,2],[1,3],[5,2]], k = 5

Output:

2

Explanation:

We can choose the following pairs: - (0,1): Because we have (1 XOR 4) + (2 XOR 2) = 5. - (2,3): Because we have (1 XOR 5) + (3 XOR 2) = 5.

Example 2:

Input:

coordinates = [[1,3],[1,3],[1,3],[1,3],[1,3]], k = 0

Output:

10

Explanation:

Any two chosen pairs will have a distance of 0. There are 10 ways to choose two pairs.

Constraints:

2 <= coordinates.length <= 50000

0 <= x

i

, y

i

<= 10

6

0 <= k <= 100

Code Snippets

C++:

```
class Solution {
public:
int countPairs(vector<vector<int>>& coordinates, int k) {

}
};
```

Java:

```
class Solution {
public int countPairs(List<List<Integer>> coordinates, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[][]} coordinates
* @param {number} k
```

```
* @return {number}
*/
var countPairs = function(coordinates, k) {

};
```

TypeScript:

```
function countPairs(coordinates: number[][], k: number): number {

};
```

C#:

```
public class Solution {
public int CountPairs(IList<IList<int>> coordinates, int k) {

}
}
```

C:

```
int countPairs(int** coordinates, int coordinatesSize, int*
coordinatesColSize, int k) {

}
```

Go:

```
func countPairs(coordinates [][]int, k int) int {

}
```

Kotlin:

```
class Solution {
fun countPairs(coordinates: List<List<Int>>, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func countPairs(_ coordinates: [[Int]], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_pairs(coordinates: Vec<Vec<i32>>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} coordinates
# @param {Integer} k
# @return {Integer}
def count_pairs(coordinates, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $coordinates
* @param Integer $k
* @return Integer
*/
function countPairs($coordinates, $k) {

}
}
```

Dart:

```
class Solution {
int countPairs(List<List<int>> coordinates, int k) {

}
```

```
}
```

Scala:

```
object Solution {
def countPairs(coordinates: List[List[Int]], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_pairs(coordinates :: [[integer]], k :: integer) :: integer
def count_pairs(coordinates, k) do

end
end
```

Erlang:

```
-spec count_pairs(Coordinates :: [[integer()]], K :: integer()) -> integer().
count_pairs(Coordinates, K) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Count Pairs of Points With Distance k
* 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<vector<int>>& coordinates, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Count Pairs of Points With Distance k
* 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(List<List<Integer>> coordinates, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Pairs of Points With Distance k
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, coordinates: List[List[int]], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Pairs of Points With Distance k
* 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[][]} coordinates
* @param {number} k
* @return {number}
*/
var countPairs = function(coordinates, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Pairs of Points With Distance k
```

```
* 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(coordinates: number[][], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Count Pairs of Points With Distance k
* 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(IList<IList<int>> coordinates, int k) {

}
}
```

C Solution:

```
/*
* Problem: Count Pairs of Points With Distance k
* 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** coordinates, int coordinatesSize, int*
coordinatesColSize, int k) {

}
```

Go Solution:

```
// Problem: Count Pairs of Points With Distance k
// 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(coordinates [][]int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countPairs(coordinates: List<List<Int>>, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func countPairs(_ coordinates: [[Int]], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Pairs of Points With Distance k
// 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(coordinates: Vec<Vec<i32>>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} coordinates
# @param {Integer} k
# @return {Integer}
def count_pairs(coordinates, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $coordinates
* @param Integer $k
* @return Integer
*/
function countPairs($coordinates, $k) {

}
}
```

Dart Solution:

```
class Solution {
int countPairs(List<List<int>> coordinates, int k) {

}
}
```

Scala Solution:

```
object Solution {
def countPairs(coordinates: List[List[Int]], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_pairs(coordinates :: [[integer]], k :: integer) :: integer
def count_pairs(coordinates, k) do

end
end
```

Erlang Solution:

```
-spec count_pairs(Coordinates :: [[integer()]], K :: integer()) -> integer().
count_pairs(Coordinates, K) ->
.
```

Racket Solution:

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