

Problem 2354: Number of Excellent Pairs

Problem Information

Difficulty: **Hard**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a

0-indexed

positive integer array

nums

and a positive integer

k

.

A pair of numbers

(num1, num2)

is called

excellent

if the following conditions are satisfied:

Both

the numbers

num1

and

num2

exist in the array

nums

.

The sum of the number of set bits in

num1 OR num2

and

num1 AND num2

is greater than or equal to

k

, where

OR

is the bitwise

OR

operation and

AND

is the bitwise

AND

operation.

Return

the number of

distinct

excellent pairs

.

Two pairs

(a, b)

and

(c, d)

are considered distinct if either

$a \neq c$

or

$b \neq d$

. For example,

(1, 2)

and

(2, 1)

are distinct.

Note

that a pair

(num1, num2)

such that

num1 == num2

can also be excellent if you have at least

one

occurrence of

num1

in the array.

Example 1:

Input:

nums = [1,2,3,1], k = 3

Output:

5

Explanation:

The excellent pairs are the following: - (3, 3). (3 AND 3) and (3 OR 3) are both equal to (11) in binary. The total number of set bits is $2 + 2 = 4$, which is greater than or equal to $k = 3$. - (2, 3) and (3, 2). (2 AND 3) is equal to (10) in binary, and (2 OR 3) is equal to (11) in binary. The total number of set bits is $1 + 2 = 3$. - (1, 3) and (3, 1). (1 AND 3) is equal to (01) in binary, and (1 OR 3) is equal to (11) in binary. The total number of set bits is $1 + 2 = 3$. So the number of

excellent pairs is 5.

Example 2:

Input:

nums = [5,1,1], k = 10

Output:

0

Explanation:

There are no excellent pairs for this array.

Constraints:

$1 \leq \text{nums.length} \leq 10$

5

$1 \leq \text{nums}[i] \leq 10$

9

$1 \leq k \leq 60$

Code Snippets

C++:

```
class Solution {
public:
    long long countExcellentPairs(vector<int>& nums, int k) {

    }
};
```

Java:

```
class Solution {  
    public long countExcellentPairs(int[] nums, int k) {  
  
    }  
}
```

Python3:

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

Python:

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

JavaScript:

```
/**  
 * @param {number[]} nums  
 * @param {number} k  
 * @return {number}  
 */  
var countExcellentPairs = function(nums, k) {  
  
};
```

TypeScript:

```
function countExcellentPairs(nums: number[], k: number): number {  
  
};
```

C#:

```

public class Solution {
    public long CountExcellentPairs(int[] nums, int k) {

    }
}

```

C:

```

long long countExcellentPairs(int* nums, int numsSize, int k) {

}

```

Go:

```

func countExcellentPairs(nums []int, k int) int64 {

}

```

Kotlin:

```

class Solution {
    fun countExcellentPairs(nums: IntArray, k: Int): Long {

    }
}

```

Swift:

```

class Solution {
    func countExcellentPairs(_ nums: [Int], _ k: Int) -> Int {

    }
}

```

Rust:

```

impl Solution {
    pub fn count_excellent_pairs(nums: Vec<i32>, k: i32) -> i64 {

    }
}

```

Ruby:

```

# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def count_excellent_pairs(nums, k)

end

```

PHP:

```

class Solution {

    /**
     * @param Integer[] $nums
     * @param Integer $k
     * @return Integer
     */
    function countExcellentPairs($nums, $k) {

    }

}

```

Dart:

```

class Solution {
  int countExcellentPairs(List<int> nums, int k) {

  }

}

```

Scala:

```

object Solution {
  def countExcellentPairs(nums: Array[Int], k: Int): Long = {

  }

}

```

Elixir:

```

defmodule Solution do

  @spec count_excellent_pairs(nums :: [integer], k :: integer) :: integer
  def count_excellent_pairs(nums, k) do

  end

end

```

```
end
end
```

Erlang:

```
-spec count_excellent_pairs(Nums :: [integer()], K :: integer()) ->
integer().
count_excellent_pairs(Nums, K) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
 * Problem: Number of Excellent Pairs
 * Difficulty: Hard
 * Tags: array, hash, search
 *
 * 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:
    long long countExcellentPairs(vector<int>& nums, int k) {

    }
};
```

Java Solution:

```

/**
 * Problem: Number of Excellent Pairs
 * Difficulty: Hard
 * Tags: array, hash, search
 *
 * 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 long countExcellentPairs(int[] nums, int k) {

}
}

```

Python3 Solution:

```

"""
Problem: Number of Excellent Pairs
Difficulty: Hard
Tags: array, hash, search

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 countExcellentPairs(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

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

```

JavaScript Solution:

```
/**
 * Problem: Number of Excellent Pairs
 * Difficulty: Hard
 * Tags: array, hash, search
 *
 * 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
 * @param {number} k
 * @return {number}
 */
var countExcellentPairs = function(nums, k) {

};
```

TypeScript Solution:

```
/**
 * Problem: Number of Excellent Pairs
 * Difficulty: Hard
 * Tags: array, hash, search
 *
 * 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 countExcellentPairs(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
 * Problem: Number of Excellent Pairs
 * Difficulty: Hard
```

```

* Tags: array, hash, search
*
* 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 long CountExcellentPairs(int[] nums, int k) {

}
}

```

C Solution:

```

/*
* Problem: Number of Excellent Pairs
* Difficulty: Hard
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

long long countExcellentPairs(int* nums, int numsSize, int k) {

}

```

Go Solution:

```

// Problem: Number of Excellent Pairs
// Difficulty: Hard
// Tags: array, hash, search
//
// 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 countExcellentPairs(nums []int, k int) int64 {

```

```
}
```

Kotlin Solution:

```
class Solution {  
    fun countExcellentPairs(nums: IntArray, k: Int): Long {  
  
    }  
}
```

Swift Solution:

```
class Solution {  
    func countExcellentPairs(_ nums: [Int], _ k: Int) -> Int {  
  
    }  
}
```

Rust Solution:

```
// Problem: Number of Excellent Pairs  
// Difficulty: Hard  
// Tags: array, hash, search  
//  
// 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_excellent_pairs(nums: Vec<i32>, k: i32) -> i64 {  
  
    }  
}
```

Ruby Solution:

```
# @param {Integer[]} nums  
# @param {Integer} k  
# @return {Integer}  
def count_excellent_pairs(nums, k)
```

```
end
```

PHP Solution:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums  
     * @param Integer $k  
     * @return Integer  
     */  
    function countExcellentPairs($nums, $k) {  
  
    }  
}
```

Dart Solution:

```
class Solution {  
    int countExcellentPairs(List<int> nums, int k) {  
  
    }  
}
```

Scala Solution:

```
object Solution {  
    def countExcellentPairs(nums: Array[Int], k: Int): Long = {  
  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
    @spec count_excellent_pairs(nums :: [integer], k :: integer) :: integer  
    def count_excellent_pairs(nums, k) do  
  
    end  
end
```

Erlang Solution:

```
-spec count_excellent_pairs(Nums :: [integer()], K :: integer()) ->
integer().
count_excellent_pairs(Nums, K) ->
.
```

Racket Solution:

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