Problem 1735: Count Ways to Make Array With Product

Problem Information

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

Problem Description

You are given a 2D integer array,

queries

. For each

queries[i]

, where

queries[i] = [n

i

, k

i

]

, find the number of different ways you can place positive integers into an array of size

n

i

such that the product of the integers is

k

i

. As the number of ways may be too large, the answer to the

i

th

query is the number of ways

modulo

10

9

$+ 7$

.

Return

an integer array

answer

where

answer.length == queries.length

, and

answer[i]

is the answer to the

i

th

query.

Example 1:

Input:

queries = [[2,6],[5,1],[73,660]]

Output:

[4,1,50734910]

Explanation:

Each query is independent. [2,6]: There are 4 ways to fill an array of size 2 that multiply to 6: [1,6], [2,3], [3,2], [6,1]. [5,1]: There is 1 way to fill an array of size 5 that multiply to 1: [1,1,1,1,1]. [73,660]: There are 1050734917 ways to fill an array of size 73 that multiply to 660. 1050734917 modulo 10

9

+ 7 = 50734910.

Example 2:

Input:

queries = [[1,1],[2,2],[3,3],[4,4],[5,5]]

Output:

[1,2,3,10,5]

Constraints:

1 <= queries.length <= 10

4

1 <= n

i

, k

i

<= 10

4

Code Snippets

C++:

```
class Solution {
public:
vector<int> waysToFillArray(vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public int[] waysToFillArray(int[][] queries) {

}
}
```

Python3:

```
class Solution:
def waysToFillArray(self, queries: List[List[int]]) -> List[int]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function waysToFillArray(queries: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] WaysToFillArray(int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* waysToFillArray(int** queries, int queriesSize, int* queriesColSize,
int* returnSize) {
```

```
}
```

Go:

```
func waysToFillArray(queries [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun waysToFillArray(queries: Array<IntArray>): IntArray {

}
}
```

Swift:

```
class Solution {
func waysToFillArray(_ queries: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn ways_to_fill_array(queries: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} queries
# @return {Integer[]}
def ways_to_fill_array(queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $queries
* @return Integer[]
*/
function waysToFillArray($queries) {

}
}
```

Dart:

```
class Solution {
List<int> waysToFillArray(List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def waysToFillArray(queries: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec ways_to_fill_array(queries :: [[integer]]) :: [integer]
def ways_to_fill_array(queries) do

end
end
```

Erlang:

```
-spec ways_to_fill_array(Queries :: [[integer()]]) -> [integer()].
ways_to_fill_array(Queries) ->
.
```

Racket:

```
(define/contract (ways-to-fill-array queries)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Ways to Make Array With Product
* Difficulty: Hard
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
vector<int> waysToFillArray(vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Count Ways to Make Array With Product
* Difficulty: Hard
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int[] waysToFillArray(int[][] queries) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Count Ways to Make Array With Product
Difficulty: Hard
Tags: array, dp, math

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def waysToFillArray(self, queries: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Ways to Make Array With Product
* Difficulty: Hard
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Count Ways to Make Array With Product
* Difficulty: Hard
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function waysToFillArray(queries: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Count Ways to Make Array With Product
* Difficulty: Hard
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int[] WaysToFillArray(int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Count Ways to Make Array With Product
* Difficulty: Hard
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* waysToFillArray(int** queries, int queriesSize, int* queriesColSize,
int* returnSize) {

}
```

Go Solution:

```
// Problem: Count Ways to Make Array With Product
// Difficulty: Hard
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func waysToFillArray(queries [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun waysToFillArray(queries: Array<IntArray>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func waysToFillArray(_ queries: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Count Ways to Make Array With Product
// Difficulty: Hard
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn ways_to_fill_array(queries: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} queries
# @return {Integer[]}
def ways_to_fill_array(queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $queries
* @return Integer[]
*/
function waysToFillArray($queries) {
```

```
}
}
```

Dart Solution:

```
class Solution {
List<int> waysToFillArray(List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def waysToFillArray(queries: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec ways_to_fill_array(queries :: [[integer]]) :: [integer]
def ways_to_fill_array(queries) do

end
end
```

Erlang Solution:

```
-spec ways_to_fill_array(Queries :: [[integer()]]) -> [integer()].
ways_to_fill_array(Queries) ->
.
```

Racket Solution:

```
(define/contract (ways-to-fill-array queries)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```