Problem 1269: Number of Ways to Stay in the Same Place After Some Steps

Problem Information

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

Problem Description

You have a pointer at index

0

in an array of size

arrLen

. At each step, you can move 1 position to the left, 1 position to the right in the array, or stay in the same place (The pointer should not be placed outside the array at any time).

Given two integers

steps

and

arrLen

, return the number of ways such that your pointer is still at index

0

after

exactly

steps

steps. Since the answer may be too large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

steps = 3, arrLen = 2

Output:

4

Explanation:

There are 4 differents ways to stay at index 0 after 3 steps. Right, Left, Stay Stay, Right, Left Right, Stay, Left Stay, Stay, Stay

Example 2:

Input:

steps = 2, arrLen = 4

Output:

2

Explanation:

There are 2 differents ways to stay at index 0 after 2 steps Right, Left Stay, Stay

Example 3:

Input:

steps = 4, arrLen = 2

Output:

8

Constraints:

1 <= steps <= 500

1 <= arrLen <= 10

6

Code Snippets

C++:

```
class Solution {
public:
int numWays(int steps, int arrLen) {

}
};
```

Java:

```
class Solution {
public int numWays(int steps, int arrLen) {
```

```
}
}
```

Python3:

```
class Solution:
def numWays(self, steps: int, arrLen: int) -> int:
```

Python:

```
class Solution(object):
def numWays(self, steps, arrLen):
"""
:type steps: int
:type arrLen: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} steps
* @param {number} arrLen
* @return {number}
*/
var numWays = function(steps, arrLen) {

};
```

TypeScript:

```
function numWays(steps: number, arrLen: number): number {

};
```

C#:

```
public class Solution {
public int NumWays(int steps, int arrLen) {

}
```

```
}
```

C:

```
int numWays(int steps, int arrLen) {

}
```

Go:

```
func numWays(steps int, arrLen int) int {

}
```

Kotlin:

```
class Solution {
fun numWays(steps: Int, arrLen: Int): Int {

}
}
```

Swift:

```
class Solution {
func numWays(_ steps: Int, _ arrLen: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_ways(steps: i32, arr_len: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} steps
# @param {Integer} arr_len
```

```
# @return {Integer}
def num_ways(steps, arr_len)

end
```

PHP:

```
class Solution {

/**
* @param Integer $steps
* @param Integer $arrLen
* @return Integer
*/
function numWays($steps, $arrLen) {

}
}
```

Dart:

```
class Solution {
int numWays(int steps, int arrLen) {

}
}
```

Scala:

```
object Solution {
def numWays(steps: Int, arrLen: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_ways(steps :: integer, arr_len :: integer) :: integer
def num_ways(steps, arr_len) do

end
```

```
end
```

Erlang:

```
-spec num_ways(Steps :: integer(), ArrLen :: integer()) -> integer().
num_ways(Steps, ArrLen) ->
.
```

Racket:

```
(define/contract (num-ways steps arrLen)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Ways to Stay in the Same Place After Some Steps
* Difficulty: Hard
* Tags: array, dp
*
* 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 numWays(int steps, int arrLen) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Ways to Stay in the Same Place After Some Steps
* Difficulty: Hard
```

```
* Tags: array, dp
*
* 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 numWays(int steps, int arrLen) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Ways to Stay in the Same Place After Some Steps
Difficulty: Hard
Tags: array, dp

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 numWays(self, steps: int, arrLen: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numWays(self, steps, arrLen):
"""
:type steps: int
:type arrLen: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Ways to Stay in the Same Place After Some Steps
* Difficulty: Hard
* Tags: array, dp
*
* 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} steps
* @param {number} arrLen
* @return {number}
*/
var numWays = function(steps, arrLen) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Ways to Stay in the Same Place After Some Steps
* Difficulty: Hard
* Tags: array, dp
*
* 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 numWays(steps: number, arrLen: number): number {

};
```

C# Solution:

```
/*
* Problem: Number of Ways to Stay in the Same Place After Some Steps
* Difficulty: Hard
* Tags: array, dp
*
```

```
* 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 NumWays(int steps, int arrLen) {

}
}
```

C Solution:

```
/*
* Problem: Number of Ways to Stay in the Same Place After Some Steps
* Difficulty: Hard
* Tags: array, dp
*
* 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
*/

int numWays(int steps, int arrLen) {

}
```

Go Solution:

```
// Problem: Number of Ways to Stay in the Same Place After Some Steps
// Difficulty: Hard
// Tags: array, dp
//
// 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 numWays(steps int, arrLen int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numWays(steps: Int, arrLen: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func numWays(_ steps: Int, _ arrLen: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Ways to Stay in the Same Place After Some Steps
// Difficulty: Hard
// Tags: array, dp
//
// 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 num_ways(steps: i32, arr_len: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} steps
# @param {Integer} arr_len
# @return {Integer}
def num_ways(steps, arr_len)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $steps
* @param Integer $arrLen
* @return Integer
*/
function numWays($steps, $arrLen) {

}
}
```

Dart Solution:

```
class Solution {
int numWays(int steps, int arrLen) {

}
}
```

Scala Solution:

```
object Solution {
def numWays(steps: Int, arrLen: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_ways(steps :: integer, arr_len :: integer) :: integer
def num_ways(steps, arr_len) do

end
end
```

Erlang Solution:

```
-spec num_ways(Steps :: integer(), ArrLen :: integer()) -> integer().
num_ways(Steps, ArrLen) ->
.
```

Racket Solution:

```
(define/contract (num-ways steps arrLen)
(-> exact-integer? exact-integer? exact-integer?)
)
```