Problem 2400: Number of Ways to Reach a Position After Exactly k Steps

Problem Information

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

Problem Description

You are given two

positive

integers

startPos

and

endPos

. Initially, you are standing at position

startPos

on an

infinite

number line. With one step, you can move either one position to the left, or one position to the right.

Given a positive integer

k

, return

the number of

different

ways to reach the position

endPos

starting from

startPos

, such that you perform

exactly

k

steps

. Since the answer may be very large, return it

modulo

10

9

+ 7

.

Two ways are considered different if the order of the steps made is not exactly the same.

Note

that the number line includes negative integers.

Example 1:

Input:

startPos = 1, endPos = 2, k = 3

Output:

3

Explanation:

We can reach position 2 from 1 in exactly 3 steps in three ways: - 1 -> 2 -> 3 -> 2. - 1 -> 2 -> 1 -> 2. - 1 -> 0 -> 1 -> 2. It can be proven that no other way is possible, so we return 3.

Example 2:

Input:

startPos = 2, endPos = 5, k = 10

Output:

0

Explanation:

It is impossible to reach position 5 from position 2 in exactly 10 steps.

Constraints:

1 <= startPos, endPos, k <= 1000

Code Snippets

C++:

```
class Solution {
public:
int numberOfWays(int startPos, int endPos, int k) {

}
};
```

Java:

```
class Solution {
public int numberOfWays(int startPos, int endPos, int k) {

}
}
```

Python3:

```
class Solution:
def numberOfWays(self, startPos: int, endPos: int, k: int) -> int:
```

Python:

```
class Solution(object):
def numberOfWays(self, startPos, endPos, k):
"""
:type startPos: int
:type endPos: int
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} startPos
* @param {number} endPos
* @param {number} k
* @return {number}
*/
var numberOfWays = function(startPos, endPos, k) {
```

```
};
```

TypeScript:

```
function numberOfWays(startPos: number, endPos: number, k: number): number {

};
```

C#:

```
public class Solution {
public int NumberOfWays(int startPos, int endPos, int k) {

}
}
```

C:

```
int numberOfWays(int startPos, int endPos, int k) {

}
```

Go:

```
func numberOfWays(startPos int, endPos int, k int) int {

}
```

Kotlin:

```
class Solution {
fun numberOfWays(startPos: Int, endPos: Int, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func numberOfWays(_ startPos: Int, _ endPos: Int, _ k: Int) -> Int {

}
```

```
}
```

Rust:

```
impl Solution {
pub fn number_of_ways(start_pos: i32, end_pos: i32, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} start_pos
# @param {Integer} end_pos
# @param {Integer} k
# @return {Integer}
def number_of_ways(start_pos, end_pos, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $startPos
* @param Integer $endPos
* @param Integer $k
* @return Integer
*/
function numberOfWays($startPos, $endPos, $k) {

}
}
```

Dart:

```
class Solution {
int numberOfWays(int startPos, int endPos, int k) {

}
}
```

Scala:

```
object Solution {
def numberOfWays(startPos: Int, endPos: Int, k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec number_of_ways(start_pos :: integer, end_pos :: integer, k :: integer)
:: integer
def number_of_ways(start_pos, end_pos, k) do

end
end
```

Erlang:

```
-spec number_of_ways(StartPos :: integer(), EndPos :: integer(), K ::
integer()) -> integer().
number_of_ways(StartPos, EndPos, K) ->
.
```

Racket:

```
(define/contract (number-of-ways startPos endPos k)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Ways to Reach a Position After Exactly k Steps
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
```

```
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int numberOfWays(int startPos, int endPos, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Ways to Reach a Position After Exactly k Steps
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int numberOfWays(int startPos, int endPos, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Ways to Reach a Position After Exactly k Steps
Difficulty: Medium
Tags: dp, math

Approach: Dynamic programming with memoization or tabulation
Time Complexity: O(n * m) where n and m are problem dimensions
Space Complexity: O(n) or O(n * m) for DP table
"""
```

```
class Solution:
def numberOfWays(self, startPos: int, endPos: int, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numberOfWays(self, startPos, endPos, k):
"""
:type startPos: int
:type endPos: int
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Ways to Reach a Position After Exactly k Steps
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} startPos
* @param {number} endPos
* @param {number} k
* @return {number}
*/
var numberOfWays = function(startPos, endPos, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Ways to Reach a Position After Exactly k Steps
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

function numberOfWays(startPos: number, endPos: number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Number of Ways to Reach a Position After Exactly k Steps
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int NumberOfWays(int startPos, int endPos, int k) {

}
}
```

C Solution:

```
/*
* Problem: Number of Ways to Reach a Position After Exactly k Steps
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

int numberOfWays(int startPos, int endPos, int k) {

}
```

Go Solution:

```
// Problem: Number of Ways to Reach a Position After Exactly k Steps
// Difficulty: Medium
// Tags: dp, math
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

func numberOfWays(startPos int, endPos int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numberOfWays(startPos: Int, endPos: Int, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func numberOfWays(_ startPos: Int, _ endPos: Int, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Ways to Reach a Position After Exactly k Steps
// Difficulty: Medium
// Tags: dp, math
```

```
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn number_of_ways(start_pos: i32, end_pos: i32, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} start_pos
# @param {Integer} end_pos
# @param {Integer} k
# @return {Integer}
def number_of_ways(start_pos, end_pos, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $startPos
* @param Integer $endPos
* @param Integer $k
* @return Integer
*/
function numberOfWays($startPos, $endPos, $k) {

}
}
```

Dart Solution:

```
class Solution {
int numberOfWays(int startPos, int endPos, int k) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def numberOfWays(startPos: Int, endPos: Int, k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec number_of_ways(start_pos :: integer, end_pos :: integer, k :: integer)
:: integer
def number_of_ways(start_pos, end_pos, k) do

end
end
```

Erlang Solution:

```
-spec number_of_ways(StartPos :: integer(), EndPos :: integer(), K ::
integer()) -> integer().
number_of_ways(StartPos, EndPos, K) ->
.
```

Racket Solution:

```
(define/contract (number-of-ways startPos endPos k)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```