Problem 3154: Find Number of Ways to Reach the K-th Stair

Problem Information

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

Problem Description

You are given a

non-negative

integer

k

. There exists a staircase with an infinite number of stairs, with the

lowest

stair numbered 0.

Alice has an integer

jump

, with an initial value of 0. She starts on stair 1 and wants to reach stair

k

using

any

number of

operations

. If she is on stair

i

, in one

operation

she can:

Go down to stair

i - 1

. This operation

cannot

be used consecutively or on stair 0.

Go up to stair

i + 2

jump

. And then,

jump

becomes

jump + 1

.

Return the

total

number of ways Alice can reach stair

k

.

Note

that it is possible that Alice reaches the stair

k

, and performs some operations to reach the stair

k

again.

Example 1:

Input:

k = 0

Output:

2

Explanation:

The 2 possible ways of reaching stair 0 are:

Alice starts at stair 1.

Using an operation of the first type, she goes down 1 stair to reach stair 0.

Alice starts at stair 1.

Using an operation of the first type, she goes down 1 stair to reach stair 0.

Using an operation of the second type, she goes up 2

0

stairs to reach stair 1.

Using an operation of the first type, she goes down 1 stair to reach stair 0.

Example 2:

Input:

k = 1

Output:

4

Explanation:

The 4 possible ways of reaching stair 1 are:

Alice starts at stair 1. Alice is at stair 1.

Alice starts at stair 1.

Using an operation of the first type, she goes down 1 stair to reach stair 0.

Using an operation of the second type, she goes up 2

0

stairs to reach stair 1.

Alice starts at stair 1.

Using an operation of the second type, she goes up 2

0

stairs to reach stair 2.

Using an operation of the first type, she goes down 1 stair to reach stair 1.

Alice starts at stair 1.

Using an operation of the first type, she goes down 1 stair to reach stair 0.

Using an operation of the second type, she goes up 2

0

stairs to reach stair 1.

Using an operation of the first type, she goes down 1 stair to reach stair 0.

Using an operation of the second type, she goes up 2

1

stairs to reach stair 2.

Using an operation of the first type, she goes down 1 stair to reach stair 1.

Constraints:

0 <= k <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int waysToReachStair(int k) {

}
};
```

Java:

```
class Solution {
public int waysToReachStair(int k) {

}
}
```

Python3:

```
class Solution:
def waysToReachStair(self, k: int) -> int:
```

Python:

```
class Solution(object):
def waysToReachStair(self, k):
"""
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} k
* @return {number}
*/
var waysToReachStair = function(k) {

};
```

}

TypeScript:

```
function waysToReachStair(k: number): number {

};
```

C#:

```
public class Solution {
public int WaysToReachStair(int k) {

}
}
```

C:

```
int waysToReachStair(int k) {

}
```

Go:

```
func waysToReachStair(k int) int {

}
```

Kotlin:

```
class Solution {
fun waysToReachStair(k: Int): Int {

}
}
```

Swift:

```
class Solution {
func waysToReachStair(_ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn ways_to_reach_stair(k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} k
# @return {Integer}
def ways_to_reach_stair(k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $k
* @return Integer
*/
function waysToReachStair($k) {

}
}
```

Dart:

```
class Solution {
int waysToReachStair(int k) {

}
}
```

Scala:

```
object Solution {
def waysToReachStair(k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec ways_to_reach_stair(k :: integer) :: integer
def ways_to_reach_stair(k) do

end
end
```

Erlang:

```
-spec ways_to_reach_stair(K :: integer()) -> integer().
ways_to_reach_stair(K) ->
.
```

Racket:

```
(define/contract (ways-to-reach-stair k)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Number of Ways to Reach the K-th Stair
* Difficulty: Hard
* 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 waysToReachStair(int k) {

}
};
```

Java Solution:

```
/**
* Problem: Find Number of Ways to Reach the K-th Stair
* Difficulty: Hard
* 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 waysToReachStair(int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Number of Ways to Reach the K-th Stair
Difficulty: Hard
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 waysToReachStair(self, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def waysToReachStair(self, k):
"""
:type k: int
:rtype: int
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Find Number of Ways to Reach the K-th Stair
* Difficulty: Hard
* 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} k
* @return {number}
*/
var waysToReachStair = function(k) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Number of Ways to Reach the K-th Stair
* Difficulty: Hard
* 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 waysToReachStair(k: number): number {

};
```

C# Solution:

```
/*
* Problem: Find Number of Ways to Reach the K-th Stair
* Difficulty: Hard
* 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 WaysToReachStair(int k) {

}
}
```

C Solution:

```
/*
* Problem: Find Number of Ways to Reach the K-th Stair
* Difficulty: Hard
* 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 waysToReachStair(int k) {

}
```

Go Solution:

```
// Problem: Find Number of Ways to Reach the K-th Stair
// Difficulty: Hard
// 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 waysToReachStair(k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun waysToReachStair(k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func waysToReachStair(_ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find Number of Ways to Reach the K-th Stair
// Difficulty: Hard
// 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 ways_to_reach_stair(k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} k
# @return {Integer}
def ways_to_reach_stair(k)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $k
* @return Integer
*/
function waysToReachStair($k) {

}
}
```

Dart Solution:

```
class Solution {
int waysToReachStair(int k) {

}
}
```

Scala Solution:

```
object Solution {
def waysToReachStair(k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec ways_to_reach_stair(k :: integer) :: integer
def ways_to_reach_stair(k) do

end
end
```

Erlang Solution:

```
-spec ways_to_reach_stair(K :: integer()) -> integer().
ways_to_reach_stair(K) ->
.
```

Racket Solution:

```
(define/contract (ways-to-reach-stair k)
(-> exact-integer? exact-integer?)
)
```