Problem 818: Race Car

Problem Information

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

Problem Description

Your car starts at position

0

and speed

+1

on an infinite number line. Your car can go into negative positions. Your car drives automatically according to a sequence of instructions

'A'

(accelerate) and

'R'

(reverse):

When you get an instruction

'A'

, your car does the following:

position += speed

speed *= 2

When you get an instruction

'R'

, your car does the following:

If your speed is positive then

speed = -1

otherwise

speed = 1

Your position stays the same.

For example, after commands

"AAR"

, your car goes to positions

0 --> 1 --> 3 --> 3

, and your speed goes to

1 --> 2 --> 4 --> -1

.

Given a target position

target

, return

the length of the shortest sequence of instructions to get there

.

Example 1:

Input:

target = 3

Output:

2

Explanation:

The shortest instruction sequence is "AA". Your position goes from 0 --> 1 --> 3.

Example 2:

Input:

target = 6

Output:

5

Explanation:

The shortest instruction sequence is "AAARA". Your position goes from 0 --> 1 --> 3 --> 7 --> 7 --> 6.

Constraints:

1 <= target <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int racecar(int target) {

}
};
```

Java:

```
class Solution {
public int racecar(int target) {

}
}
```

Python3:

```
class Solution:
def racecar(self, target: int) -> int:
```

Python:

```
class Solution(object):
def racecar(self, target):
"""
:type target: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} target
* @return {number}
*/
var racecar = function(target) {

};
```

TypeScript:

```
function racecar(target: number): number {

};
```

C#:

```
public class Solution {
public int Racecar(int target) {

}
}
```

C:

```
int racecar(int target) {

}
```

Go:

```
func racecar(target int) int {

}
```

Kotlin:

```
class Solution {
fun racecar(target: Int): Int {

}
}
```

Swift:

```
class Solution {
func racecar(_ target: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn racecar(target: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} target
# @return {Integer}
def racecar(target)

end
```

PHP:

```
class Solution {

/**
* @param Integer $target
* @return Integer
*/
function racecar($target) {

}
}
```

Dart:

```
class Solution {
int racecar(int target) {

}
}
```

Scala:

```
object Solution {
def racecar(target: Int): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec racecar(target :: integer) :: integer
def racecar(target) do

end
end
```

Erlang:

```
-spec racecar(Target :: integer()) -> integer().
racecar(Target) ->
.
```

Racket:

```
(define/contract (racecar target)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Race Car
* Difficulty: Hard
* Tags: dp
*
* 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 racecar(int target) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Race Car
* Difficulty: Hard
* Tags: dp
*
* 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 racecar(int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Race Car
Difficulty: Hard
Tags: dp

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 racecar(self, target: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def racecar(self, target):
"""
:type target: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Race Car
* Difficulty: Hard
* Tags: dp
*
* 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} target
* @return {number}
*/
var racecar = function(target) {

};
```

TypeScript Solution:

```
/**
* Problem: Race Car
* Difficulty: Hard
* Tags: dp
*
* 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 racecar(target: number): number {

};
```

C# Solution:

```
/*
* Problem: Race Car
* Difficulty: Hard
* Tags: dp
*
* 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 Racecar(int target) {

}
}
```

C Solution:

```
/*
* Problem: Race Car
* Difficulty: Hard
* Tags: dp
*
* 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 racecar(int target) {

}
```

Go Solution:

```
// Problem: Race Car
// Difficulty: Hard
// Tags: dp
//
// 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 racecar(target int) int {

}
```

Kotlin Solution:

```
class Solution {
fun racecar(target: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func racecar(_ target: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Race Car
// Difficulty: Hard
// Tags: dp
//
// 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 racecar(target: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} target
# @return {Integer}
def racecar(target)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $target
* @return Integer
*/
function racecar($target) {

}
}
```

Dart Solution:

```
class Solution {
int racecar(int target) {

}
}
```

Scala Solution:

```
object Solution {
def racecar(target: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec racecar(target :: integer) :: integer
def racecar(target) do

end
```

```
end
```

Erlang Solution:

```
-spec racecar(Target :: integer()) -> integer().
racecar(Target) ->
.
```

Racket Solution:

```
(define/contract (racecar target)
(-> exact-integer? exact-integer?)
)
```