Problem 2318: Number of Distinct Roll Sequences

Problem Information

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

Problem Description

You are given an integer

n

. You roll a fair 6-sided dice

n

times. Determine the total number of

distinct

sequences of rolls possible such that the following conditions are satisfied:

The

greatest common divisor

of any

adjacent

values in the sequence is equal to

1

.

There is

at least

a gap of

2

rolls between

equal

valued rolls. More formally, if the value of the

i

th

roll is

equal

to the value of the

j

th

roll, then

abs(i - j) > 2

.

Return

the

total number

of distinct sequences possible

. Since the answer may be very large, return it

modulo

10

9

+ 7

.

Two sequences are considered distinct if at least one element is different.

Example 1:

Input:

n = 4

Output:

184

Explanation:

Some of the possible sequences are (1, 2, 3, 4), (6, 1, 2, 3), (1, 2, 3, 1), etc. Some invalid sequences are (1, 2, 1, 3), (1, 2, 3, 6). (1, 2, 1, 3) is invalid since the first and third roll have an equal value and abs(1 - 3) = 2 (i and j are 1-indexed). (1, 2, 3, 6) is invalid since the greatest common divisor of 3 and 6 = 3. There are a total of 184 distinct sequences possible, so we return 184.

Example 2:

Input:

n = 2

Output:

22

Explanation:

Some of the possible sequences are (1, 2), (2, 1), (3, 2). Some invalid sequences are (3, 6), (2, 4) since the greatest common divisor is not equal to 1. There are a total of 22 distinct sequences possible, so we return 22.

Constraints:

1 <= n <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int distinctSequences(int n) {

}
};
```

Java:

```
class Solution {
public int distinctSequences(int n) {

}
}
```

Python3:

```
class Solution:
def distinctSequences(self, n: int) -> int:
```

Python:

```
class Solution(object):
def distinctSequences(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {number}
*/
var distinctSequences = function(n) {

};
```

TypeScript:

```
function distinctSequences(n: number): number {

};
```

C#:

```
public class Solution {
public int DistinctSequences(int n) {

}
}
```

C:

```
int distinctSequences(int n) {

}
```

Go:

```
func distinctSequences(n int) int {

}
```

Kotlin:

```
class Solution {
fun distinctSequences(n: Int): Int {

}
}
```

Swift:

```
class Solution {
func distinctSequences(_ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn distinct_sequences(n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer}
def distinct_sequences(n)

end
```

PHP:

```
class Solution {

/**
```

```
* @param Integer $n
* @return Integer
*/
function distinctSequences($n) {

}
}
```

Dart:

```
class Solution {
int distinctSequences(int n) {

}
}
```

Scala:

```
object Solution {
def distinctSequences(n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec distinct_sequences(n :: integer) :: integer
def distinct_sequences(n) do

end
end
```

Erlang:

```
-spec distinct_sequences(N :: integer()) -> integer().
distinct_sequences(N) ->
.
```

Racket:

```
(define/contract (distinct-sequences n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Distinct Roll Sequences
* 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 distinctSequences(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Distinct Roll Sequences
* 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 distinctSequences(int n) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Number of Distinct Roll Sequences
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 distinctSequences(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def distinctSequences(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Distinct Roll Sequences
* 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} n
* @return {number}
*/
var distinctSequences = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Distinct Roll Sequences
* 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 distinctSequences(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Number of Distinct Roll Sequences
* 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 DistinctSequences(int n) {

}
}
```

C Solution:

```
/*
* Problem: Number of Distinct Roll Sequences
* 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 distinctSequences(int n) {

}
```

Go Solution:

```
// Problem: Number of Distinct Roll Sequences
// 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 distinctSequences(n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun distinctSequences(n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func distinctSequences(_ n: Int) -> Int {
```

```
}
}
```

Rust Solution:

```
// Problem: Number of Distinct Roll Sequences
// 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 distinct_sequences(n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer}
def distinct_sequences(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function distinctSequences($n) {

}
}
```

Dart Solution:

```
class Solution {
int distinctSequences(int n) {

}
}
```

Scala Solution:

```
object Solution {
def distinctSequences(n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec distinct_sequences(n :: integer) :: integer
def distinct_sequences(n) do

end
end
```

Erlang Solution:

```
-spec distinct_sequences(N :: integer()) -> integer().
distinct_sequences(N) ->
.
```

Racket Solution:

```
(define/contract (distinct-sequences n)
(-> exact-integer? exact-integer?)
)
```