Problem 1155: Number of Dice Rolls With Target Sum

Problem Information

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

Problem Description

You have

n

dice, and each dice has

k

faces numbered from

1

to

k

.

Given three integers

n

,

k

, and

target

, return

the number of possible ways (out of the

k

n

total ways)

to roll the dice, so the sum of the face-up numbers equals

target

. Since the answer may be too large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

n = 1, k = 6, target = 3

Output:

1

Explanation:

You throw one die with 6 faces. There is only one way to get a sum of 3.

Example 2:

Input:

n = 2, k = 6, target = 7

Output:

6

Explanation:

You throw two dice, each with 6 faces. There are 6 ways to get a sum of 7: 1+6, 2+5, 3+4, 4+3, 5+2, 6+1.

Example 3:

Input:

n = 30, k = 30, target = 500

Output:

222616187

Explanation:

The answer must be returned modulo 10

9

+ 7.

Constraints:

1 <= n, k <= 30

1 <= target <= 1000

Code Snippets

C++:

```
class Solution {
public:
int numRollsToTarget(int n, int k, int target) {

}
};
```

Java:

```
class Solution {
public int numRollsToTarget(int n, int k, int target) {

}
}
```

Python3:

```
class Solution:
def numRollsToTarget(self, n: int, k: int, target: int) -> int:
```

Python:

```
class Solution(object):
def numRollsToTarget(self, n, k, target):
"""
:type n: int
:type k: int
:type target: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} k
* @param {number} target
* @return {number}
*/
var numRollsToTarget = function(n, k, target) {

};
```

TypeScript:

```
function numRollsToTarget(n: number, k: number, target: number): number {

};
```

C#:

```
public class Solution {
public int NumRollsToTarget(int n, int k, int target) {

}
}
```

C:

```
int numRollsToTarget(int n, int k, int target) {

}
```

Go:

```
func numRollsToTarget(n int, k int, target int) int {

}
```

Kotlin:

```
class Solution {
fun numRollsToTarget(n: Int, k: Int, target: Int): Int {
```

```
}
}
```

Swift:

```
class Solution {
func numRollsToTarget(_ n: Int, _ k: Int, _ target: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_rolls_to_target(n: i32, k: i32, target: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} k
# @param {Integer} target
# @return {Integer}
def num_rolls_to_target(n, k, target)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @param Integer $target
* @return Integer
*/
function numRollsToTarget($n, $k, $target) {

}
```

```
}
```

Dart:

```
class Solution {
int numRollsToTarget(int n, int k, int target) {

}
}
```

Scala:

```
object Solution {
def numRollsToTarget(n: Int, k: Int, target: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_rolls_to_target(n :: integer, k :: integer, target :: integer) ::
integer
def num_rolls_to_target(n, k, target) do

end
end
```

Erlang:

```
-spec num_rolls_to_target(N :: integer(), K :: integer(), Target ::
integer()) -> integer().
num_rolls_to_target(N, K, Target) ->
.
```

Racket:

```
(define/contract (num-rolls-to-target n k target)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Dice Rolls With Target Sum
* Difficulty: Medium
* 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 numRollsToTarget(int n, int k, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Dice Rolls With Target Sum
* Difficulty: Medium
* 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 numRollsToTarget(int n, int k, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Dice Rolls With Target Sum
```

```
Difficulty: Medium
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 numRollsToTarget(self, n: int, k: int, target: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numRollsToTarget(self, n, k, target):
"""
:type n: int
:type k: int
:type target: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Dice Rolls With Target Sum
* Difficulty: Medium
* 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
* @param {number} k
* @param {number} target
* @return {number}
```

```
*/
var numRollsToTarget = function(n, k, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Dice Rolls With Target Sum
* Difficulty: Medium
* 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 numRollsToTarget(n: number, k: number, target: number): number {

};
```

C# Solution:

```
/*
* Problem: Number of Dice Rolls With Target Sum
* Difficulty: Medium
* 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 NumRollsToTarget(int n, int k, int target) {

}
}
```

C Solution:

```
/*
* Problem: Number of Dice Rolls With Target Sum
* Difficulty: Medium
* 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 numRollsToTarget(int n, int k, int target) {

}
```

Go Solution:

```
// Problem: Number of Dice Rolls With Target Sum
// Difficulty: Medium
// 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 numRollsToTarget(n int, k int, target int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numRollsToTarget(n: Int, k: Int, target: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func numRollsToTarget(_ n: Int, _ k: Int, _ target: Int) -> Int {

}
```

```
}
```

Rust Solution:

```
// Problem: Number of Dice Rolls With Target Sum
// Difficulty: Medium
// 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 num_rolls_to_target(n: i32, k: i32, target: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} k
# @param {Integer} target
# @return {Integer}
def num_rolls_to_target(n, k, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @param Integer $target
* @return Integer
*/
function numRollsToTarget($n, $k, $target) {

}
```

```
}
```

Dart Solution:

```
class Solution {
int numRollsToTarget(int n, int k, int target) {

}
}
```

Scala Solution:

```
object Solution {
def numRollsToTarget(n: Int, k: Int, target: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_rolls_to_target(n :: integer, k :: integer, target :: integer) ::
integer
def num_rolls_to_target(n, k, target) do

end
end
```

Erlang Solution:

```
-spec num_rolls_to_target(N :: integer(), K :: integer(), Target ::
integer()) -> integer().
num_rolls_to_target(N, K, Target) ->
.
```

Racket Solution:

```
(define/contract (num-rolls-to-target n k target)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```