Problem 3183: The Number of Ways to Make the Sum

Problem Information

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

Problem Description

You have an

infinite

number of coins with values 1, 2, and 6, and

only

2 coins with value 4.

Given an integer

n

, return the number of ways to make the sum of

n

with the coins you have.

Since the answer may be very large, return it

modulo

10

9

+ 7

.

Note

that the order of the coins doesn't matter and

[2, 2, 3]

is the same as

[2, 3, 2]

.

Example 1:

Input:

n = 4

Output:

4

Explanation:

Here are the four combinations:

[1, 1, 1, 1]

,

[1, 1, 2]

,

[2, 2]

,

[4]

.

Example 2:

Input:

n = 12

Output:

22

Explanation:

Note that

[4, 4, 4]

is

not

a valid combination since we cannot use 4 three times.

Example 3:

Input:

n = 5

Output:

4

Explanation:

Here are the four combinations:

[1, 1, 1, 1, 1]

,

[1, 1, 1, 2]

,

[1, 2, 2]

,

[1, 4]

.

Constraints:

1 <= n <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

```
}
}
```

C:

```
int numberOfWays(int n) {

}
```

Go:

```
func numberOfWays(n int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (number-of-ways n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: The Number of Ways to Make the Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int numberOfWays(int n) {

}
};
```

Java Solution:

```
/**
* Problem: The Number of Ways to Make the Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int numberOfWays(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: The Number of Ways to Make the Sum
Difficulty: Medium
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: The Number of Ways to Make the Sum
* Difficulty: Medium
```

```
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: The Number of Ways to Make the Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function numberOfWays(n: number): number {

};
```

C# Solution:

```
/*
* Problem: The Number of Ways to Make the Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

public class Solution {
public int NumberOfWays(int n) {

}
}
```

C Solution:

```
/*
* Problem: The Number of Ways to Make the Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int numberOfWays(int n) {

}
```

Go Solution:

```
// Problem: The Number of Ways to Make the Sum
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func numberOfWays(n int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: The Number of Ways to Make the Sum
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (number-of-ways n)
(-> exact-integer? exact-integer?)
)
```