Problem 1411: Number of Ways to Paint N × 3 Grid

Problem Information

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

Problem Description

You have a

grid

of size

n x 3

and you want to paint each cell of the grid with exactly one of the three colors:

Red

,

Yellow,

or

Green

while making sure that no two adjacent cells have the same color (i.e., no two cells that share vertical or horizontal sides have the same color).

Given

n

the number of rows of the grid, return

the number of ways

you can paint this

grid

. As the answer may grow large, the answer

must be

computed modulo

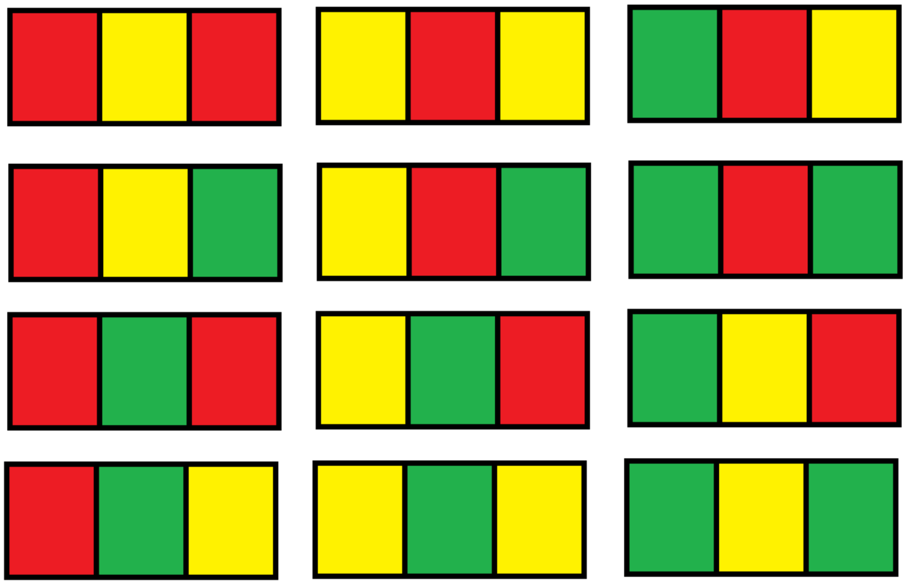
10

9

+ 7

.

Example 1:



Input:

n = 1

Output:

12

Explanation:

There are 12 possible way to paint the grid as shown.

Example 2:

Input:

n = 5000

Output:

30228214

Constraints:

n == grid.length

1 <= n <= 5000

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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


}
}
```

C:

```
int numOfWays(int n) {

}
```

Go:

```
func numOfWays(n int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Number of Ways to Paint N × 3 Grid
* 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 numOfWays(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Ways to Paint N × 3 Grid
* 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 numOfWays(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Ways to Paint N × 3 Grid
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 numOfWays(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Ways to Paint N × 3 Grid
* 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 numOfWays = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Ways to Paint N × 3 Grid
* 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 numOfWays(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Number of Ways to Paint N × 3 Grid
* 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 NumOfWays(int n) {

}
}
```

C Solution:

```
/*
* Problem: Number of Ways to Paint N × 3 Grid
* 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 numOfWays(int n) {

}
```

Go Solution:

```
// Problem: Number of Ways to Paint N × 3 Grid
// 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 numOfWays(n int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Ways to Paint N × 3 Grid
// 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 num_of_ways(n: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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