Problem 1931: Painting a Grid With Three Different Colors

Problem Information

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

Problem Description

You are given two integers

m

and

n

. Consider an

m x n

grid where each cell is initially white. You can paint each cell

red

,

green

, or

blue

. All cells

must

be painted.

Return

the number of ways to color the grid with

no two adjacent cells having the same color

. Since the answer can be very large, return it

modulo

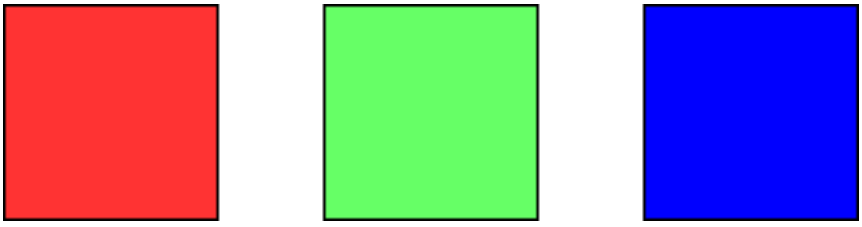
10

9

+ 7

.

Example 1:



Input:

m = 1, n = 1

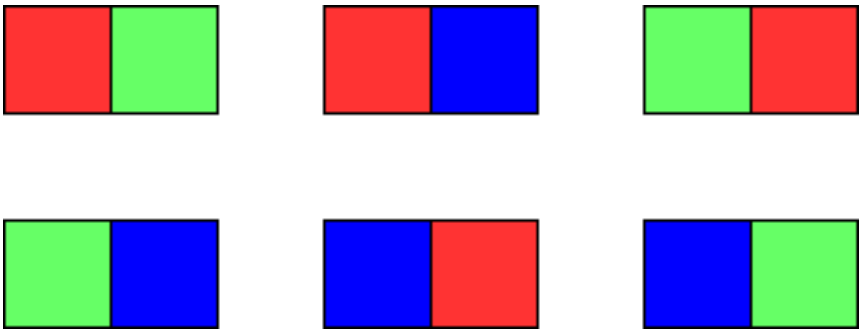
Output:

3

Explanation:

The three possible colorings are shown in the image above.

Example 2:



Input:

m = 1, n = 2

Output:

6

Explanation:

The six possible colorings are shown in the image above.

Example 3:

Input:

m = 5, n = 5

Output:

580986

Constraints:

1 <= m <= 5

1 <= n <= 1000

Code Snippets

C++:

```
class Solution {
public:
int colorTheGrid(int m, int n) {

}
};
```

Java:

```
class Solution {
public int colorTheGrid(int m, int n) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number} m
* @param {number} n
* @return {number}
*/
```

```
var colorTheGrid = function(m, n) {

};
```

TypeScript:

```
function colorTheGrid(m: number, n: number): number {

};
```

C#:

```
public class Solution {
public int ColorTheGrid(int m, int n) {

}
}
```

C:

```
int colorTheGrid(int m, int n) {

}
```

Go:

```
func colorTheGrid(m int, n int) int {

}
```

Kotlin:

```
class Solution {
fun colorTheGrid(m: Int, n: Int): Int {

}
}
```

Swift:

```
class Solution {
func colorTheGrid(_ m: Int, _ n: Int) -> Int {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn color_the_grid(m: i32, n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} m
# @param {Integer} n
# @return {Integer}
def color_the_grid(m, n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $m
* @param Integer $n
* @return Integer
*/
function colorTheGrid($m, $n) {

}
}
```

Dart:

```
class Solution {
int colorTheGrid(int m, int n) {

}
}
```

Scala:

```
object Solution {
def colorTheGrid(m: Int, n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec color_the_grid(m :: integer, n :: integer) :: integer
def color_the_grid(m, n) do

end
end
```

Erlang:

```
-spec color_the_grid(M :: integer(), N :: integer()) -> integer().
color_the_grid(M, N) ->
.
```

Racket:

```
(define/contract (color-the-grid m n)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Painting a Grid With Three Different Colors
* 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 colorTheGrid(int m, int n) {

}
};
```

Java Solution:

```
/**
* Problem: Painting a Grid With Three Different Colors
* 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 colorTheGrid(int m, int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Painting a Grid With Three Different Colors
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 colorTheGrid(self, m: int, n: int) -> int:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Painting a Grid With Three Different Colors
* 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} m
* @param {number} n
* @return {number}
*/
var colorTheGrid = function(m, n) {

};
```

TypeScript Solution:

```
/**
* Problem: Painting a Grid With Three Different Colors
* 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 colorTheGrid(m: number, n: number): number {

};
```

C# Solution:

```
/*
* Problem: Painting a Grid With Three Different Colors
* 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 ColorTheGrid(int m, int n) {

}
}
```

C Solution:

```
/*
* Problem: Painting a Grid With Three Different Colors
* 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 colorTheGrid(int m, int n) {
```

```
}
```

Go Solution:

```
// Problem: Painting a Grid With Three Different Colors
// 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 colorTheGrid(m int, n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun colorTheGrid(m: Int, n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func colorTheGrid(_ m: Int, _ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Painting a Grid With Three Different Colors
// 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 color_the_grid(m: i32, n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} m
# @param {Integer} n
# @return {Integer}
def color_the_grid(m, n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $m
* @param Integer $n
* @return Integer
*/
function colorTheGrid($m, $n) {

}
}
```

Dart Solution:

```
class Solution {
int colorTheGrid(int m, int n) {

}
}
```

Scala Solution:

```
object Solution {
def colorTheGrid(m: Int, n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec color_the_grid(m :: integer, n :: integer) :: integer
def color_the_grid(m, n) do

end
end
```

Erlang Solution:

```
-spec color_the_grid(M :: integer(), N :: integer()) -> integer().
color_the_grid(M, N) ->
.
```

Racket Solution:

```
(define/contract (color-the-grid m n)
(-> exact-integer? exact-integer? exact-integer?)
)
```