Problem 276: Paint Fence

Problem Information

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

Problem Description

You are painting a fence of

n

posts with

k

different colors. You must paint the posts following these rules:

Every post must be painted

exactly one

color.

There

cannot

be three or more

consecutive

posts with the same color.

Given the two integers

n

and

k

, return

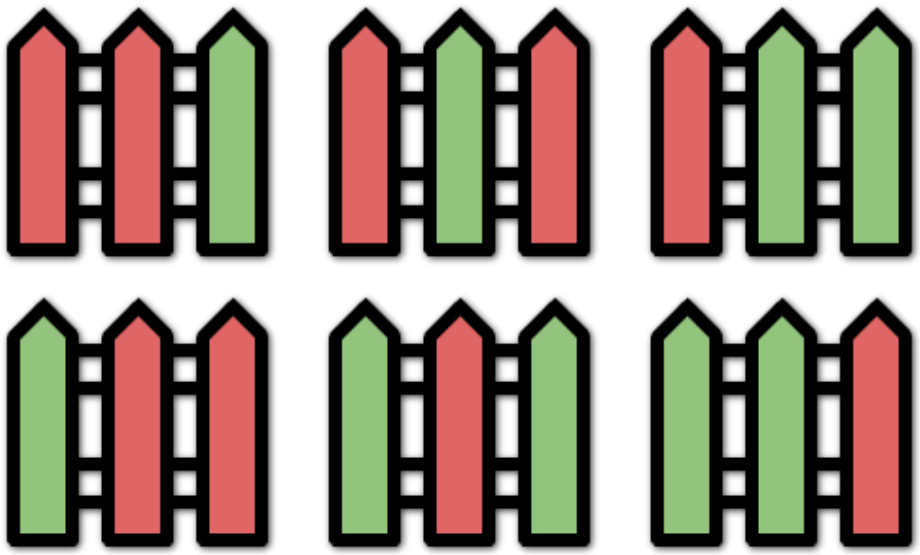
the

number of ways

you can paint the fence

.

Example 1:



Input:

n = 3, k = 2

Output:

6

Explanation:

All the possibilities are shown. Note that painting all the posts red or all the posts green is invalid because there cannot be three posts in a row with the same color.

Example 2:

Input:

n = 1, k = 1

Output:

1

Example 3:

Input:

n = 7, k = 2

Output:

42

Constraints:

1 <= n <= 50

1 <= k <= 10

5

The testcases are generated such that the answer is in the range

[0, 2

31

- 1]

for the given

n

and

k

.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int numWays(int n, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function numWays(n: number, k: number): number {

};
```

C#:

```
public class Solution {
public int NumWays(int n, int k) {

}
}
```

C:

```
int numWays(int n, int k) {

}
```

Go:

```
func numWays(n int, k int) int {
```

```
}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
```

```
* @return Integer
*/
function numWays($n, $k) {

}
}
```

Dart:

```
class Solution {
int numWays(int n, int k) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

```
)
```

Solutions

C++ Solution:

```
/*
* Problem: Paint Fence
* 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 numWays(int n, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Paint Fence
* 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 numWays(int n, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Paint Fence
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 numWays(self, n: int, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Paint Fence
* 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
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Paint Fence
* 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 numWays(n: number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Paint Fence
* 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 NumWays(int n, int k) {

}
}
```

C Solution:

```
/*
* Problem: Paint Fence
* 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 numWays(int n, int k) {

}
```

Go Solution:

```
// Problem: Paint Fence
// 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 numWays(n int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
```

```
}
```

Rust Solution:

```
// Problem: Paint Fence
// 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_ways(n: i32, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int numWays(int n, int k) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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