Problem 790: Domino and Tromino Tiling

Problem Information

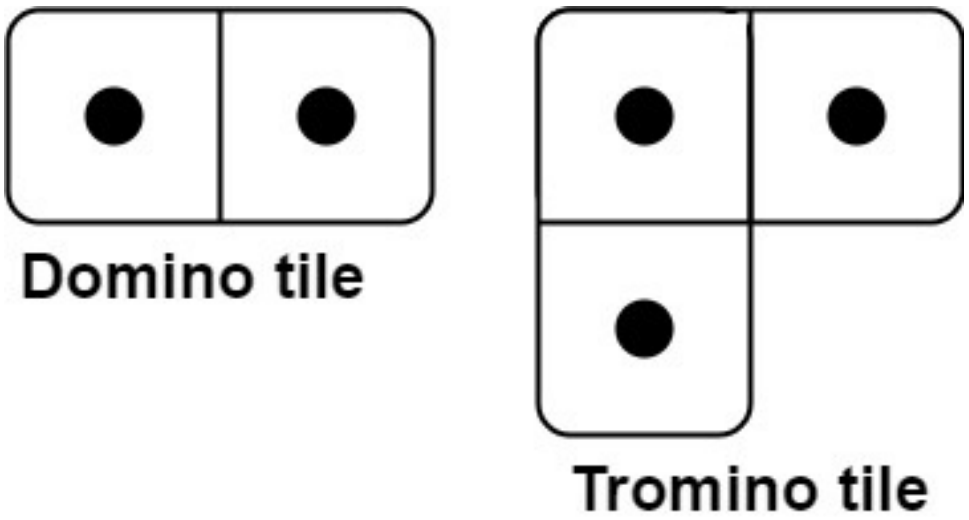
Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You have two types of tiles: a

2 x 1

domino shape and a tromino shape. You may rotate these shapes.



Given an integer n, return

the number of ways to tile an

2 x n

board

. Since the answer may be very large, return it

modulo

10

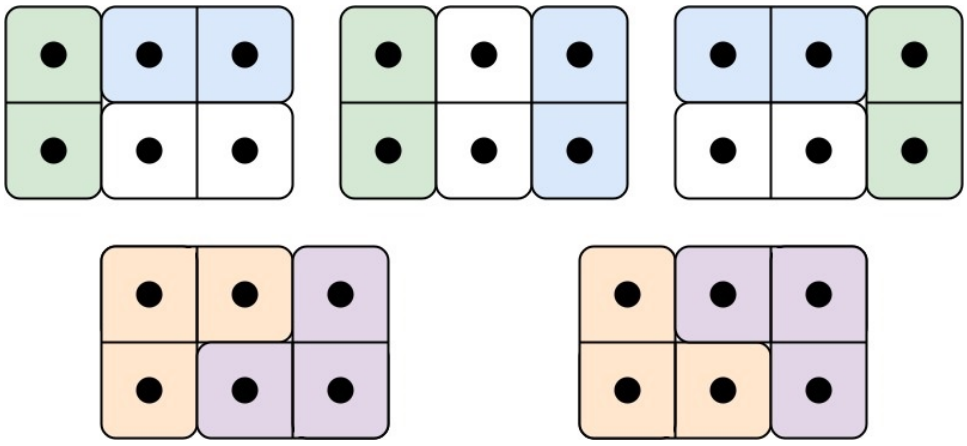
9

+ 7

.

In a tiling, every square must be covered by a tile. Two tilings are different if and only if there are two 4-directionally adjacent cells on the board such that exactly one of the tilings has both squares occupied by a tile.

Example 1:



Input:

n = 3

Output:

5

Explanation:

The five different ways are shown above.

Example 2:

Input:

n = 1

Output:

1

Constraints:

1 <= n <= 1000

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int numTilings(int n) {

}
```

Go:

```
func numTilings(n int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
```

```
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Domino and Tromino Tiling
* 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 numTilings(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Domino and Tromino Tiling
* 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 numTilings(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Domino and Tromino Tiling
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 numTilings(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Domino and Tromino Tiling
* 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
* @return {number}
*/
var numTilings = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Domino and Tromino Tiling
* 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 numTilings(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Domino and Tromino Tiling
* 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 NumTilings(int n) {

}
}
```

C Solution:

```
/*
* Problem: Domino and Tromino Tiling
* 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 numTilings(int n) {

}
```

Go Solution:

```
// Problem: Domino and Tromino Tiling
// 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 numTilings(n int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Domino and Tromino Tiling
// 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_tilings(n: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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