


Domino and Tromino Tiling

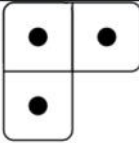
790. Domino and Tromino Tiling Solved

Medium Topics Companies

You have two types of tiles: a 2×1 domino shape and a tromino shape. You may rotate these shapes.



Domino tile

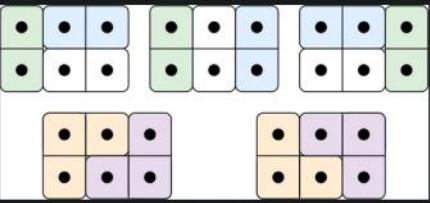


Tromino tile

Given an integer n , return the number of ways to tile an $2 \times n$ 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 tiles 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:

- $1 \leq n \leq 1000$

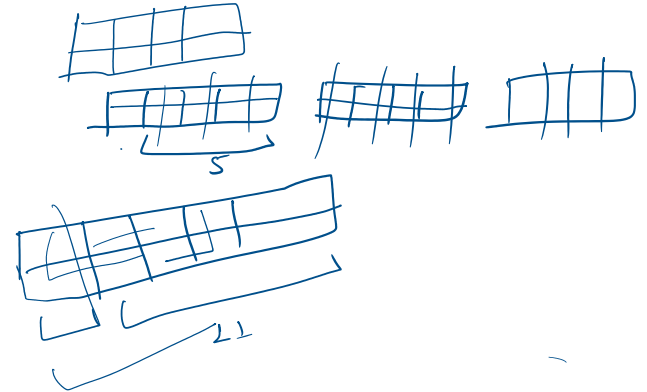
num(3)

$$dp[1] = 1$$

$$dp[2] = 2$$

$$dp[3] = 5$$

dp(3)



Approach #1: Recursion + Memoization

class Solution:

int mod = $10^9 + 7$;

public:

int numTilings(int n) {

vector<vector<int>> dp(n+1, vector<int>(n+1, -1));

return help(n, n, dp); }

// maintaining $|r1 - r2| < 1$

int help(int r1, int r2, vector<vector<int>>& dp) {

if (r1 == 0 && r2 == 0) return 1;

if (r1 <= 0 || r2 <= 0) return 0;

if (dp[r1][r2] != -1) return dp[r1][r2];

if (r1 == r2) {

dp[r1][r2] = help(r1-1, r2-1, dp) % mod;

dp[r1][r2] = (dp[r1][r2] + help(r1-2, r2-2, dp)) % mod;

dp[r1][r2] = (dp[r1][r2] + help(r1-1, r2-2, dp)) % mod;

```

    }
    dp[r1][r2] = (dp[r1][r2] + help(r1-2, r2-2, dp)) % mod;
    dp[r1][r2] = (dp[r1][r2] + help(r1-1, r2-2, dp)) % mod;
    dp[r1][r2] = (dp[r1][r2] + help(r1-2, r2-1, dp)) % mod;
}
else if (r1 > r2) {
    dp[r1][r2] = help(r1-2, r2, dp) % mod;
    dp[r1][r2] = (dp[r1][r2] + help(r1-2, r2-1, dp)) % mod;
}
else {
    dp[r1][r2] = help(r1, r2-2, dp) % mod;
    dp[r1][r2] = (dp[r1][r2] + help(r1-1, r2-2, dp)) % mod;
}
return dp[r1][r2];
}
};

```

```

C++ v Auto
1 class Solution {
2     int mod=1e9+7;
3 public:
4     int numTilings(int n) {
5         vector<vector<int>> dp(n+1, vector<int>(n+1, -1));
6         return help(n, n, dp);
7     }
8
9     // maintaining |r1-r2|<1
10
11     int help(int r1, int r2, vector<vector<int>> &dp){
12         if(r1==0 && r2==0) return 1;
13         if(r1<0 || r2<0) return 0;
14         if(dp[r1][r2]!=-1) return dp[r1][r2];
15         if(r1==r2){
16             dp[r1][r2]=help(r1-1, r2-1, dp)%mod;
17             dp[r1][r2]=(dp[r1][r2]+help(r1-2, r2-2, dp))%mod;
18             dp[r1][r2]=(dp[r1][r2]+help(r1-1, r2-2, dp))%mod;
19             dp[r1][r2]=(dp[r1][r2]+help(r1-2, r2-1, dp))%mod;
20         }
21         else if(r1>r2){
22             dp[r1][r2]=help(r1-2, r2, dp)%mod;
23             dp[r1][r2]=(dp[r1][r2]+help(r1-2, r2-1, dp))%mod;
24         }
25         else{
26             dp[r1][r2]=help(r1, r2-2, dp)%mod;
27             dp[r1][r2]=(dp[r1][r2]+help(r1-1, r2-2, dp))%mod;
28         }
29         return dp[r1][r2];
30     }
31 };

```

S.C. $O(n^2)$

T.C. $O(n)$

why $O(n)$ T.C.

Max. difference b/w $r1$ & $r2 = 1$

$|r1 - r2| \leq 1$ (always)

because of how we handle $r1$ & $r2$ 3-cases:-

because of how we handle $r1$ & $r2$ cases.

For any tile \Rightarrow width_{max} = 2 (horizontal)

We are putting horizontal in only $r1$ where $r1 > r2$.

Tabulation:- Let $dp[0] = 1$ (for identity)

$N=1$

0	1

$dp[0] = 1$

$dp[1] = 1$

$N=2$

0	1	2

$dp[1] + dp[0] =$

$dp[2] = 2$

$N=3$

0	1	2	3

$dp[2] + dp[1] = + dp[0]$

\uparrow
2

\uparrow
1

\uparrow
2

$dp[3] = 5$

$N=4$

0	1	2	3	4

$$dp[4] = dp[3] + dp[2] + dp[1] + dp[0]$$

\uparrow
5

\uparrow
2

\uparrow
2

\uparrow
2

$= 11$

$N=5$

0	1	2	3	4	5

$$dp[4] + dp[3] + dp[2] + dp[1]$$

These will continue for future N

$$\therefore dp[N] = dp[N-1] + dp[N-2] + dp[N-3] \times 2 + dp[N-4] \times 2 \dots$$

$$dp[N] = dp[N-1] + dp[N-2] + 2 \times (dp[N-3] + dp[N-4] \dots)$$

$$= dp[N-1] + (dp[N-2] + dp[N-3]) + dp[N-3] + 2 \times (dp[N-4] + dp[N-5] \dots)$$

①

$$\therefore dp[N-1] = dp[N-2] + dp[N-3] + 2 \times (dp[N-4] + dp[N-5] + \dots)$$

$$\Rightarrow 2 \times (dp[N-4] + dp[N-5] + \dots) = dp[N-1] - dp[N-2] - dp[N-3]$$

Now, put this in ①

$$dp[N] = dp[N-1] + (dp[N-2] + dp[N-3]) + dp[N-3] + dp[N-1] - dp[N-2] - dp[N-3]$$

$$= 2 \times dp[N-1] + dp[N-3]$$

$$\Rightarrow \begin{aligned} dp[N] &= dp[N-1] + 1 \\ dp[N] &= 2 \times dp[N-1] + dp[N-3] \end{aligned}$$

C++ Code :-

```
class Solution {
public:
    int mod = 1e9+7;

    int numTilings(int n) {
        vector<int> dp(1001);
        dp[0] = 1;
        dp[1] = 1;
        dp[2] = 2;

        for(int i=3; i<=n; i++) {
            dp[i] = (2 * dp[i-1]) % mod;
            dp[i] = (dp[i] + dp[i-3]) % mod;
        }
        return dp[n];
    }
};
```

```
C++ v Auto
1 class Solution {
2 public:
3     int mod=1e9+7;
4     int numTilings(int n) {
5         vector<int> dp(1001);
6         dp[0]=1;
7         dp[1]=1;
8         dp[2]=2;
9         for(int i=3; i<=n; i++){
10             dp[i]=(2*dp[i-1])%mod;
11             dp[i]=(dp[i]+dp[i-3])%mod;
12         }
13         return dp[n];
14     }
15 };
```

T.C. $O(n)$
S.C. $O(1)$