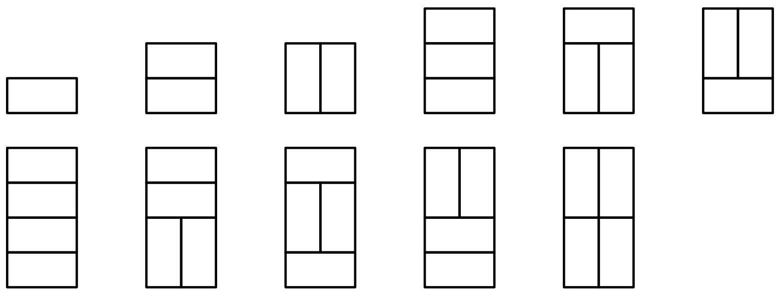
Tilings Count

Another month has passed and it is time to show how the previous Codility Certificate, codenamed Φ (Phi), can be solved. You can still give it a try, but no certificate will be granted.

Before we solve the main problem, let us consider a similar but much simpler problem. A $2 \times N$ board is given. In how many different ways can we cover it using 2×1 domino tiles? Let us denote this number by $T(N)$.

For $N = 1$ there is only one way to do it ($T(1) = 1$). For $N = 2$ there are two symmetrical ways to do it ($T(2) = 2$). For $N = 3$ it can be done in three different ways ($T(3) = 3$), but for $N = 4$ there are five (not four) different ways to cover the board ($T(4) = 5$), as shown in a figure below.



How about larger values of N? There are two cases:

- If there is a horizontal tile at the top of the board, then the number of such solutions equals $T(N - 1)$;
- If there are two vertical tiles at the top of the board, then the number of such solutions equals $T(N - 2)$.

If we fix $T(0) = 1$, we obtain:

$$\begin{aligned} T(0) &= 1 \\ T(1) &= 2 \\ T(N) &= T(N-1) + T(N-2) \end{aligned}$$

Looks familiar? Right! They are the Fibonacci numbers!

One of the fastest ways to calculate the Fibonacci numbers is to use matrix exponentiation. Let us define:

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

It is known that:

$$A^K = \begin{bmatrix} F_{K+1} & F_K \\ F_K & F_{K-1} \end{bmatrix}$$

$$A^K \begin{pmatrix} F_M \\ F_{M-1} \end{pmatrix} = \begin{pmatrix} F_{M+K} \\ F_{M+K-1} \end{pmatrix}$$

Behind the matrix exponentiation there is some useful intuition, which we will exploit in the solution to the main problem. Let us draw two horizontal lines across the board in such a way that there are K rows between the lines and L rows below the lower line. The lower L rows of the board can be covered by the domino tiles, so that no tile extends beyond them, in $T(L)$ different ways. Also, they can be covered in such a way that two tiles cross the lower line in $T(L-1)$ different ways. Similarly, the lower $L+K$ rows can be covered in $T(L+K)$ and $T(L+K-1)$ different ways respectively. We have:

$$A^K \begin{pmatrix} T(L) \\ T(L-1) \end{pmatrix} = \begin{pmatrix} T(L+K) \\ T(L+K-1) \end{pmatrix}$$

So, A^K represents how the middle K rows influence the number of ways in which the board can be covered.

Let's get back to the main problem. We deal with an $N \times M$ board (where $1 \leqslant M \leqslant 7$) and cover it with 1×1 and 2×2 tiles. Again, imagine that we cut the board with a horizontal line. The line can hit some 2×2 tiles. When merging all possible coverings of the parts of the board above and below the line, we should consider all possible configurations of the 2×2 tiles crossed by the horizontal line. Such configurations can be represented by bit vectors of length M. Since $1 \leqslant M \leqslant 7$, there are at most 128 such vectors. Of course, not all of them are feasible. If 1s represent crossed 2×2 tiles, then they must appear in sequences of even length. The following function defines the integers (from 0 to $2^M - 1$) whose binary representations are feasible bit vectors:

1: Is the binary representation feasible?

```
def feasible(x):
    ok= True
    while(x > 0):
        if x % 2 == 1:
            ok = not ok
        elif not ok:
            return False
        x = x//2
    return ok
```

Imagine that there are two horizontal lines crossing the board such that there is exactly one row of squares between them. These lines can cross some 2×2 tiles, but at each horizontal position at most one of them can cross such a tile. Simply, if one line crosses a 2×2 tile, then the other one is adjacent to its side. Let i and j be non-negative integers, such that $0 \leqslant i, j \leqslant 2^M - 1$ and their binary representations are feasible bit vectors. When is it possible that they represent positions where the two horizontal lines cross the 2×2 tiles? Only if 1s in their binary representations appear at disjoint positions. On the other hand, if it is possible, then there is only one way to do it — all free squares between the horizontal lines must be covered by 1×1 tiles. The following procedure defines a $2^M \times 2^M$ matrix A, such that $A[i][j] = 1$ if and only if i and j represent a possible combination of 2×2 tiles crossed by the two horizontal lines, and $A[i][j] = 0$ otherwise.

2: Matrix representing one row extension

```
def row_matrix(M):
    mm=2**M
    A = [[0] * mm for i in xrange(mm)]
    for i in xrange(mm):
        if feasible(i):
            for j in xrange(mm):
                if feasible(j) and (i & j) == 0:
                    A[i][j] = 1
    return A
```

If there are K rows of squares between the two horizontal lines, and i and j represent 2×2 tiles crossed by the two horizontal lines, then, in the same way as in the simpler problem analyzed at the beginning, there are $A^K[i][j]$ ways to cover the squares between the two lines. Hence, there are $A^N[0][0]$ ways to cover the board so that no 2×2 tile extends the upper or lower edge of the board. All we lack to solve the task is a matrix multiplication (modulo 10,000,007):

3: Model solution — $O(8^M \cdot \log N)$

```
def ident(size):
    Id = [[0]*size for i in xrange(size)]
    for i in xrange(size):
        Id[i][i] = 1
    return Id

def matrix_mult(A, B, ModRes):
    size = len(A)
    C = [[0]*size for i in xrange(size)]
    for i in xrange(size):
        for j in xrange(size):
            cc = 0
            for k in xrange(size):
                cc = (cc + (A[i][k] % ModRes) * (B[k][j] % ModRes)
                    )% ModRes
            C[i][j]= cc
    return C

def matrix_exp(A, N, ModRes):
    size = len(A)
    B = ident(size)
    while N > 0:
        if N % 2 == 1:
            B = matrix_mult(A, B, ModRes)
        A = matrix_mult(A, A, ModRes)
        N = N//2
    return B

def count_tilings(N, M):
    B = matrix_exp (row_matrix(M), N, 10000007)
    return B[0][0]
```

Matrix exponentiation requires $O(\log N)$ matrix multiplications. The latter ones run in cubic time. Since we deal with $2^M \times 2^M$ matrices, the overall time complexity is $O(8^M \cdot \log N)$.