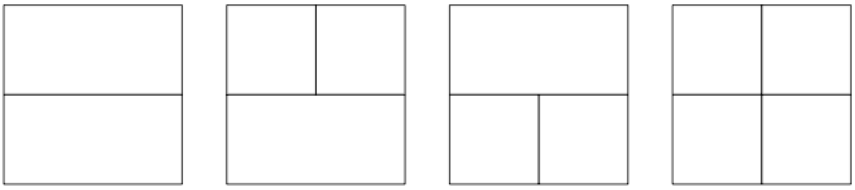
Lego Blocks

Editorial by lydxlx

By projecting the wall onto 2D plane, the question becomes how to use 1×1, 1×2, 1×3 and(or) 1×4 lego-"slice" to solidly cover a $H \times W$ rectangle without any holes or overlapping.
Please refer to the figure below for an example of the first test case, where the first three embeddings are valid, but the last is not since it is not solid.



Before solving this problem, let's consider a simpler version, which ignores the solid condition for a while. Now, the problem is decomposable, i.e., each row is an independent (same) subproblem. In other words, if we use $f[i]$ to denote # of ways to cover a $1 \times i$ bar, and let $g[i]$ be # of ways to cover a $N \times i$ rectangle ignoring the "solid" condition. Then,

$$g[i] = f[i]^N.$$

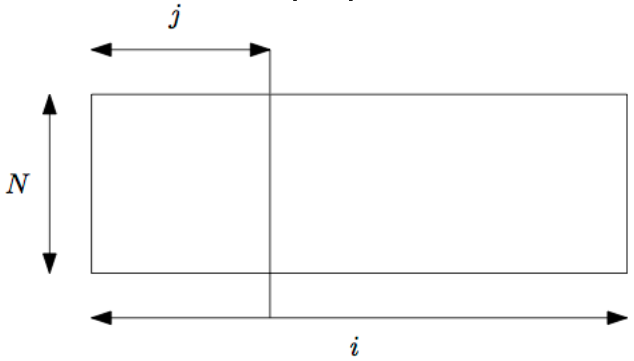
Recurrence of $f[i]$

Since there cannot be any holes for the embedding, we have to put a lego-slice, say $1 \times k$, at the very left of the bar. Then, the length of the uncovered bar becomes $i - k$, and obviously we have $f[i-k]$ different ways to cover. Therefore,

$$f[i] = \begin{cases} f[i-1] + f[i-2] + f[i-3] + f[i-4] & \text{if } i > 0 \\ 1 & \text{if } i = 0 \\ 0 & \text{if } i < 0 \end{cases}.$$

Back to the primal problem

$g[M]$ gives us # of ways to embed a $N \times M$ block without the "solid" condition. However, we can easily remove those invalid (unsolid) cases from $g[M]$. Let $h[i]$ denote # of solid embedding of a $N \times i$ retangle.
For those unsolid embedding of size $N \times i$, we enumerate all the leftmost cuts that split the wall into two parts without damaging any lego-slices. Assume the block to the left of the cut has width j w.r.t. the current cut (see the figure below), and consequently, the one to the right has width $i - j$. Since the cut we just made is assumed to be the leftmost one, the left block has to be a solid (valid) one, and hence has $h[j]$ different ways of embedding. On the other hand, there is no particular constraint on the right one, and thus $g[i-j]$ choices.



In such a case, the rule of product applies, and therefore

Statistics
Difficulty: 0.7157039711
Time Complexity: O(M
Required Knowledge: D
Publish Date: Jul 20 201

This is a Practice Challe

$$h[i] = g[i] - \sum_{j=1}^{i-1} h[j] * g[i-j].$$

The final answer will be $h[M]$, and be careful to deal with modulo when subtraction exists. :)

Problem Setter's code :

```
#include <iostream>
#include <cstring>
using namespace std;

const long long mod = 1000000007;
int t;
int n;
int m;
long long f[1111];
long long g[1111];
long long h[1111];

long long pow(long long a, int p)
{
    long long ans = 1;
    while(p)
    {
        if (p % 2) ans = ans * a % mod;
        a = a * a % mod;
        p /= 2;
    }
    return ans;
}
int main(void)
{
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);

    f[0] = 1;
    for (int i=1; i<=1000; i++)
        for (int j=1; j<=4; j++)
            if (i - j >= 0) f[i] = (f[i] + f[i-j]) % mod;

    for (cin >> t; t; t--)
    {
        cin >> n >> m;
        for (int i=1; i<=m; i++) g[i] = pow(f[i], n);

        memset(h, 0, sizeof(h));
        h[1] = 1;
        for (int i=2; i<=m; i++)
        {
            h[i] = g[i];
            long long tmp = 0;
            for (int j=1; j<i; j++)
                tmp = (tmp + h[j] * g[i-j]) % mod;
            h[i] = (h[i] - tmp + mod) % mod;
        }
        cout << h[m] << "\n";
    }

    return 0;
}
```