CSES Problem Set

Nebhrajani A.V.

February 26, 2021

Contents

1. Dynamic Programming

Some of the harder problems in dynamic programming with insightful solutions.

1.1 Counting Towers

1.1.1 Statement

Your task is to build a tower whose width is 2 and height is n. You have an unlimited supply of blocks whose width and height are integers.

Given n, how many different towers can you build? Mirrored and rotated towers are counted separately if they look different.

Input: The first input line contains an integer t: the number of tests. After this, there are t lines, and each line contains an integer n: the height of the tower.

Output: For each test, print the number of towers modulo $10^9 + 7$.

Constraints: $1 \leq t \leq 100$, $1 \leq n \leq 10^6$

Example:

Input:

3

2

6

1337

Output:

8

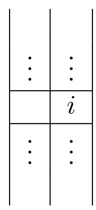
2864

640403945

1.1.2 Solution

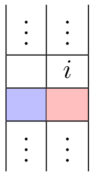
This is a particularly neat problem, in that it is tempting to write a recurrence that multiplies successive tilings at various points. However, such combinatorial logic is rarely (if ever) useful for DP.

Instead, consider 'building' the tower from a position i, somewhere in the middle of the tower.



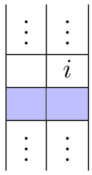
Okay, so we're somewhere in what is clearly going to be a DP sequence. When in such a position, it's always useful to analyze the *options* we have, as though we're playing a game. As in a game, our options are based off the state of the board the opponent left us with. Let us therefore consider the $i-1$ th tower row to see our options.

Clearly, it can be in any one of the following two states:



(`bool` is true)

Or,

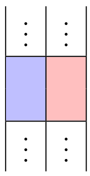


(`bool` is false)

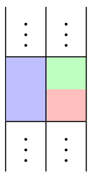
Alright. Convince yourself that $i-1$ has no other possibilities: either joint or broken. Each of these cases of $i-1$ means very different options for when we're at i, so let's define a `bool` that is true when the tiles are joined.

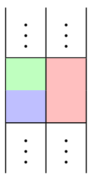
Now, in the `bool` is false case, the options are:

1. Extend both.



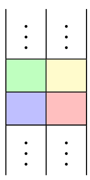
2. Close one, extend the other.



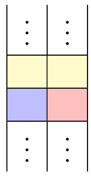


3. Close both. This presents the same two options for our current cell:

   (a) Make two tiles.

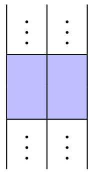


   (b) Make one fat tile.



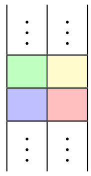
On the other hand, if `bool` is true,

1. Extend the fat tile.

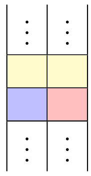


2. Close the fat tile. This presents the same two options as (3) for our current cell:

   (a) Make two tiles.



   (b) Make one fat tile.



Hopefully, you see what we're getting at here: $i+1$ has the same set of options to select from as i did, and we can define a `bool` for every successive floor in the tower.

Let us therefore define two recursions, one for the `bool` 0 case and one for the 1 case.

$$f(i,0) = f(i+1,0) + 2 \times f(i+1,0) + f(i+1,0) + f(i+1,1)$$
$$f(i,1) = f(i+1,1) + f(i+1,0) + f(i+1,1)$$

Or, combining like terms,

$$f(i,0) = 4 \times f(i+1,0) + f(i+1,1)$$
$$f(i,1) = 2 \times f(i+1,1) + f(i+1,0)$$

Let's DP this in C++.

```cpp
#include <bits/stdc++.h>

long int dp[1000001][2];
int main(void)
{
  dp[1][1] = dp[1][0] = 1;
  for (int i = 2; i < 1000001; ++i) {
    dp[i][0] = ((dp[i-1][0] * 4) + dp[i-1][1]) % 1000000007;
```

```
        dp[i][1] = ((dp[i-1][1] * 2) + dp[i-1][0]) % 1000000007;
    }

    std::cout << (dp[2][0] + dp[2][1]) % 1000000007 << "\n";
    std::cout << (dp[6][0] + dp[6][1]) % 1000000007 << "\n";
    std::cout << (dp[1337][0] + dp[1337][1]) % 1000000007 << "\n";
    return 0;
}
```

8
2864
640403945

For CSES, you'll have to take an input on `STDIN` for testcases and output `(dp[n][0] + dp[n][1]) % 1000000007` according to `n`.