Towers ★

Points: 454.76 Rank: 5577

Editorial by pkacprzak

You can also check the editorial on my blog: http://chasethered.com/?p=310

In the editorial I omit the fact that we have to count the time needed to build all towers - we will count here just the number of different towers and to get the result, you have to multiply this number by 2

Simpler problem first

There is a restriction, that every bricks is of height at most 15 and N is at most 10^{18}. Lets first try to solve a simpler problem and assume that there are only bricks of height 1 or 2 and $N \leq 10^6$.

Let f_n be the number of different towers of height n which can be build of given bricks. In our simpler problem we have:

$f_1 = 1$, because there is only one tower of height 1 consisting of one bricks of height 1

$f_2 = 2$, because there are exactly two towers of height 2 - one consisting of two bricks of height 1 and the other consisting of one brick of height 2

We can now give a recursive definition for f_n when $n > 2$:

$$f_n = f_{n-1} + f_{n-2}$$

because we can get one unique tower of height n by placing a brick of height 1 on any tower of height $n-1$ and one unique tower of height also n by placing a brick of height 2 on any tower of height $n-2$.

If we compute f_n bottom-up (like in dynamic programming) rather than using recursion explicitly, we can do it in $O(n)$ time which is fine if n is not as big as in the problem statement.

Removing heights restriction

Lets remove the first restriction, so we have to deal now with bricks of at most 15 different heights (from 1 to 15).

Let h_i indicates whether we have a brick of height i:

$$h_i = \begin{cases} 1 & \text{if we have bricks of height } i \\ 0 & \text{if we don't have bricks of height } i \end{cases}$$

then the general recursion equation is:

$f_n = 0$ for $n \leq 0$

$f_0 = 1$

$f_n = h_1 \cdot f_{n-1} + h_2 \cdot f_{n-2} + h_3 \cdot f_{n-3} + \ldots + h_15 \cdot f_{n-15}$ for $n > 0$

Solving the original problem

The last thing that we have do deal with is the really big n - up to 10^{18}. We will do it defining the recursive equation as a matrix multiplication problem and solve it using fast matrix exponentiation.

Lets assume that we've computed f_n for $n \leq 15$ using dynamic programming and let M be a 15×15 matrix defined as follows:

$$M = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ h_{15} & h_{14} & h_{13} & h_{12} & h_{11} & h_{10} & h_9 & h_8 & h_7 & h_6 & h_5 & h_4 & h_3 & h_2 & h_1 \end{bmatrix}$$

Let V be the following vector:

$$V = [f_1 \quad f_2 \quad f_3 \quad f_4 \quad f_5 \quad f_6 \quad f_7 \quad f_8 \quad f_9 \quad f_{10} \quad f_{11} \quad f_{12} \quad f_{13} \quad f_{14} \quad f_{15}]^T$$

if we multiply $M \cdot V$ we get:

$$R = [f_2 \quad f_3 \quad f_4 \quad f_5 \quad f_6 \quad f_7 \quad f_8 \quad f_9 \quad f_10 \quad f_{11} \quad f_{12} \quad f_{13} \quad f_{14} \quad f_{15} \quad f_{16}]^T$$

and we can return f_{16} from here, so in order to compute f_n for $n \leq 15$ we can compute:

$$R = \underbrace{(M \cdot (M \cdot \cdots \cdot (M}_{n-15} \cdot V)))$$

which can be written in the following form, because matrix multiplication is associative:

$$R = M^{(n-15)} \cdot V$$

and we can compute n-th power of a matrix of size $m \times m$ is $O(m^3 \cdot \log_2 n)$ using fast exponentiation by squaring and in our problem we have $m = 15$ and $n \le 10^{18}$ which is perfectly fine in terms of time limits. One speedup here can be to do exponentiation iteratively rather than recursively.

Set by pkacprzak

Problem Setter's code:

C++

```
--C++
#include <iostream>
#include <cstdio>
#include <string>
#include <vector>
#include <cmath>
#include <algorithm>
using namespace std;
#define FOR(i,n) for(int (i)=0;(i)<(n);++(i))

typedef unsigned long long ull;
typedef vector<int> vi;
typedef vector<vector<ull> > matrix;
const int MOD = 1000000000 + 7;

matrix mmul(matrix M1, matrix M2)
{
   matrix R(15);
   FOR(i, 15)
   {
      R[i].assign(15, 0);
      FOR(j, 15)
      {
         FOR(k, 15)
         {
            R[i][j] = (R[i][j] + M1[i][k] * M2[k][j]) % MOD;
         }
      }
   }
   return R;
}

matrix mpow(matrix M, ull n)
{
   if(n == 0)
   {
      matrix M1(15);
      FOR(i, 15)
      {
         M1[i].assign(15, 0);
         FOR(j, 15)
         {
            if(i == j)
               M1[i][j] = 1;
         }
      }
      return M1;
   }
   else if(n % 2 == 0)
   {
      matrix R = mpow(M, n / 2);
      return mmul(R, R);
   }
   else
   {
      matrix R = mpow(M, n - 1);
      return mmul(R, M);
   }
}

ull V[15];
```

```
int main()
{
//GENERATE M
   matrix M(15);
   FOR(i, 15)
   {
      M[i].assign(15, 0);
      FOR(j, 15)
      {
         if(i + 1 == j)
         {
            M[i][j] = 1;
         }
      }
   }
//INPUT
   int k;
   ull n;
   vi jumps;
   vi res;
   ull tmp;

   cin >> n;
   cin >> k;
   FOR(i, 15)
   {
      V[i] = 0;
   }
   M[14].assign(15, 0);
   FOR(i, k)
   {
      cin >> tmp;
      M[14][14 - tmp + 1] = 1;
   }
   FOR(i, 15)
   {
      FOR(j, i + 1)
      {
         if(M[14][14 - j] == 1)
         {
            if(i - j == 0)
            {
               V[i]++;
            }
            else
            {
               V[i] += V[i - j - 1];
            }
         }
      }
   }
   if(n <= 15)
   {
      ull res = 0;
      res = (V[n - 1] * 2) % MOD;
      cout << res << endl;
   }
   else
   {
      matrix R = mpow(M, n - 15);
      ull res = 0;
      FOR(j, 15)
      {
         res = (res + R[14][j] * V[j]) % MOD;
      }
      res = (res * 2) % MOD;
      cout << res << endl;
   }
   return 0;
}
```

Feedback

Was this editorial helpful?

Yes No