斐波那契数列

Josh-Cena

2020 年 10 月 10 日

斐波那契数列：

$$F_n = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ F_{n-2} + F_{n-1}, & n > 1 \end{cases}$$

给定 n，求 $F_n \bmod 10^9 + 7$。

数据规模	内存限制	运行时间
$0 \le n \le 10^{19}$	64 MB	1.0 s

题解. 10^{19} 显然灭掉了所有用循环解决的想法。有没有比简单的 $\mathcal{O}(n)$ 更好一点的方法？用**矩阵快速幂**，可以达到 $\mathcal{O}(\log n)$。观察到：

$$\begin{pmatrix} F_{n+1} \\ F_{n+2} \end{pmatrix} = \begin{pmatrix} F_{n+1} \\ F_n + F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix}$$

这一步对于所有递推数列都是适用的，因此在有经验之后应该非常容易得到。一般地，对于 $F_{n+2} = aF_n + bF_{n+1}$，有

$$\begin{pmatrix} F_{n+1} \\ F_{n+2} \end{pmatrix} = \begin{pmatrix} F_{n+1} \\ aF_n + bF_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ a & b \end{pmatrix} \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix}$$

从递推式中有

$$\begin{pmatrix} F_{n+m} \\ F_{n+m+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^m \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix}$$

取 $n=0$，得到

$$\begin{pmatrix} F_m \\ F_{m+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^m \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

因此把问题转化成了如何求矩阵 m 次方的问题。如果设 $m = 2^0 a_0 + 2^1 a_1 + 2^2 a_2 + \dots$（也就是把 m 用二进制表示），那么有

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^m = \left(\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^1\right)^{a_0} \times \left(\left(\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^2\right)^{a_1} \times \left(\left(\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^4\right)^{a_2} \dots\right.\right.$$

而这些矩阵的 2^k 次方，完全可以预处理。当 m 的数量级为 10^{19} 时，$k < \log_2 10^{19} < 64$，最多只需要存储 63 个矩阵。并且

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^{2^k} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^{2^{k-1}} \times \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^{2^{k-1}}$$

这些乘方可以在 $\mathcal{O}(\log m)$ 时间内得到。这便是快速幂的思想：计算所有的 2^k 次方，然后把其中需要的那些组合起来即可。

下面是 C++ 代码，其中最繁琐的部分是实现矩阵乘法：

```cpp
#include <iostream>
#include <cmath>

using namespace std;

unsigned long long mat_pow[64][4] = {
   {0,1,1,1}
};

int fib(unsigned long long k){
   unsigned long long t00 = 0, t01 = 1, t10 = 1, t11 = 1;
   for (int i = 0; i < 64; i++) {
      if (k & (1ull << i)) {
         unsigned long long a = (t00 * mat_pow[i][0] + t01 * mat_pow[i][2]) %
            1000000007;
         unsigned long long b = (t00 * mat_pow[i][1] + t01 * mat_pow[i][3]) %
            1000000007;
         unsigned long long c = (t10 * mat_pow[i][0] + t11 * mat_pow[i][2]) %
            1000000007;
         unsigned long long d = (t10 * mat_pow[i][1] + t11 * mat_pow[i][3]) %
            1000000007;
```

```
            t00 = a;
            t01 = b;
            t10 = c;
            t11 = d;
        }
    }
    return t00 % 1000000007;
}

int main(){
    for (int i = 1; i < 64; i++) {
        mat_pow[i][0] = (mat_pow[i-1][0] * mat_pow[i-1][0] + mat_pow[i-1][1] *
            mat_pow[i-1][2]) % 1000000007;
        mat_pow[i][1] = (mat_pow[i-1][0] * mat_pow[i-1][1] + mat_pow[i-1][1] *
            mat_pow[i-1][3]) % 1000000007;
        mat_pow[i][2] = (mat_pow[i-1][0] * mat_pow[i-1][2] + mat_pow[i-1][2] *
            mat_pow[i-1][3]) % 1000000007;
        mat_pow[i][3] = (mat_pow[i-1][1] * mat_pow[i-1][2] + mat_pow[i-1][3] *
            mat_pow[i-1][3]) % 1000000007;
    }
    unsigned long long n;
    cin >> n;
    cout << fib(n) << endl;
    return 0;
}
```