

EOlymp 11116. Number Series

Nama : Timothy Hosia Budianto

NRP : 50525211098

Permasalahan dan pendekatan :

Input

One integer n ($1 \leq n \leq 10^{12}$).

Output

Print one integer - the number of different numerical sequences consisting of integers with a difference of 1 and elements with a sum of n .

Soal ini pertama meminta untuk mengeluarkan Berapa banyak barisan bilangan bulat yang berbeda dimana selisih antara dua elemen yang berdekatan adalah 1 dan jumlah elemen dalam barisan tersebut adalah n ?

Jadi soal akan meminta input 1 nilai interger

Kemudian akan mengoutputkan berapa banyak jumlah barisan angka yang dapat terbuat dengan syarat nilai yang bersebelahan memiliki selisih 1, dan jumlah semua barisannya adalah n .

Abstraksi :

Untuk menyari penyelesaian, pertama tama kita harus menuliskan segala kemungkinan testcase yang ada dan melihat pola yang terbentuk. Test casenya adalah sebagai berikut.

EOlymp 11116. Number series		
n	barisan	Jumlah barisan
1	<ul style="list-style-type: none">• 1• 0, 1	2
2	<ul style="list-style-type: none">• 2• -1, 0, 1, 2	2
3	<ul style="list-style-type: none">• 3• 1, 2• 0, 1, 2• -2, -1, 0, 1, 2, 3	4
7	<ul style="list-style-type: none">• 7• 3, 4• -2, -1, 0, 1, 2, 3, 4• -6, -5, -4, -3 - 2, -1, 0, 1, 2, 3, 4, 5, 6, 7	4

Soalnya memang sedikit ambigu karna banyak kemungkinan saat Menyusun baris angka tersebut, namun harus dibandingkan dengan testcase yang sudah disediakan. Oleh karena itu kita mendapatkan syarat tambahan dari testcase sebagai berikut :

1. Barisan angka pasti ada 1 versi dimana hanya angka tersebut itu saja.
2. Barisan angka tidak ada yang terus berulang (seperti 1,0,1,0,1). Ini artinya selisih juga selalu diambil dari angka terbesar dikurang angka terkecil.
3. Urutan angka tidak bisa dibalik, dan disusun secara incremental.

EOlymp 11116. Number series		
n	Jumlah barisan	Jumlah faktor pembagi gasal
1	2	1 (1)
2	2	1 (1)
3	4	2 (1,3)
7	4	2 (1,7)

Setelah melihat syarat tambahan tersebut, kita melihat pola yang dihasilkan dari test case yang ada. Dapat dilihat bahwa hasilnya didapatkan dari

2 kali jumlah factor(pembagi) yang gasal.

Penyelesaian ini sesuai dengan teori yang ada di buku matematika diskrit edisi 8 yaitu pada bab 4 point 4.3 mengenai “Primes and Greatest Common Divisors”. Bahwa faktor pembagi ada hubungannya dengan bilangan prima. Teori ini juga membantu untuk menentukan kompleksitas tertinggi. Teorinya sebagai berikut.

EOlymp 11116. Number series

4.3.3 Trial Division

It is often important to show that a given integer is prime. For instance, in cryptography, large primes are used in some methods for making messages secret. One procedure for showing that an integer is prime is based on the following observation.

THEOREM 2 If n is a composite integer, then n has a prime divisor less than or equal to \sqrt{n} .

Proof: If n is composite, by the definition of a composite integer, we know that it has a factor a with $1 < a < n$. Hence, by the definition of a factor of a positive integer, we have $n = ab$, where b is a positive integer greater than 1. We will show that $a \leq \sqrt{n}$ or $b \leq \sqrt{n}$. If $a > \sqrt{n}$ and $b > \sqrt{n}$, then $ab > \sqrt{n} \cdot \sqrt{n} = n$, which is a contradiction. Consequently, $a \leq \sqrt{n}$ or $b \leq \sqrt{n}$. Because both a and b are divisors of n , we see that n has a positive divisor not exceeding \sqrt{n} . This divisor is either prime or, by the fundamental theorem of arithmetic, has a prime divisor less than itself. In either case, n has a prime divisor less than or equal to \sqrt{n} . \triangleleft



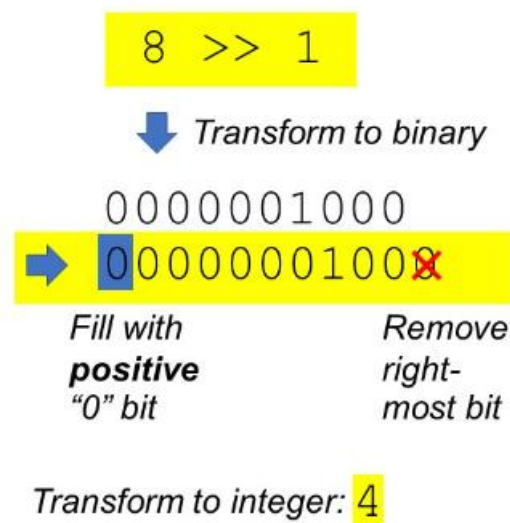
Dilihat dari teori tersebut bahwa bila n adalah bilangan composiste (bukan prima), faktor ganjil pembaginya hanya perlu di check sampai akar dari n . Contoh pada bilangan 25, kita cukup memeriksa bilangan 2 sampai 5. Karena tidak mungkin lagi ada pembagi prima lebih besar dari akar 25. Arena itulah kompleksitas waktu tertinggi nya adalah $O(\sqrt{n})$.

Implementasi :

```
*main.cpp X
1  #include<stdio.h>
2
3  int main()
4  {
5      long long n;
6      scanf("%lld", &n);
7
8      while(!(n&1)) n>>=1;
9
10
11     long long ans=2;
12
13     for(long long i = 3; i * i <= n; i+=2)
14     {
15         int fact = 1;
16         while(n%i==0) n = n / i, fact++;
17         ans = ans * fact;
18     }
19     if (n>1) ans<<=1;
20
21     printf("%lld\n", ans); //print hasil
22     return 0;
23 }
24
```

Line 5 – 6 pertama mendecclare variable long long n dan meminta input nilai interger.

Line 8 ketika nilainya tidak ganjil (dilihat dari not (n and 1), nilai 1 akan muncul jika salah satu nilainya tidak true yang menandakan bilangan ganjil karena operand and) n akan di shift right, yang berarti nilai dibagi 2. Hal tersebut dilakukan untuk menghilangkan faktor pembagi 2. Metode pembagi menggunakan metode bitwise untuk menghemat waktu.



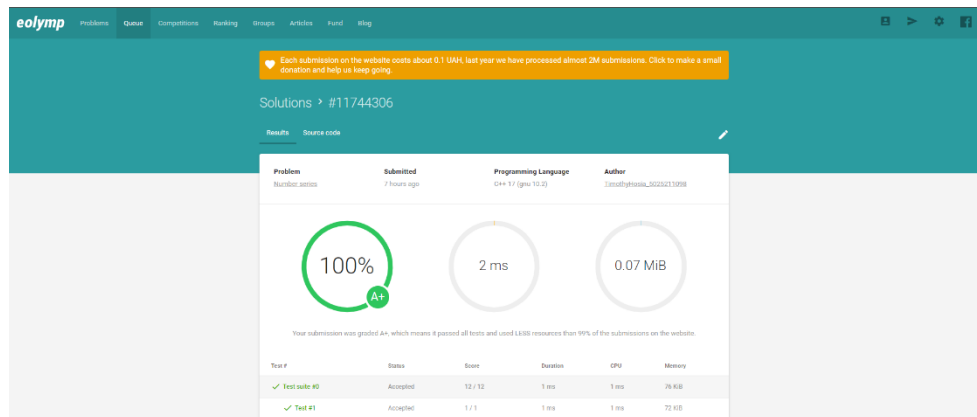
Sumber <https://blog.finxter.com/python-bitwise-right-shift-operator/>

Line 11 mengassign variable ans dengan nilai 2 agar tidak perlu lagi operasi dikali 2 untuk mendapatkan hasil.

Kemudian line 13 – 18 menggunakan perulangan loop mencari faktor pembagi gasal degan nilai mulai dari 3 dan inkrementasi plus 2 (karena ganjil). Perulangan loop dilakukan dengan batas sampai kurang dari akar n sesuai dengan penjelasan teori diatas. Di dalam loop ada variable fact yang digunakan untuk menjadi counter untuk jumlah pembagi, yang nantinya akan dikalikan 2 untuk mendapatkan hasil sesuai rumus. Fungsi while di dalam loop untuk mencari nilai faktor yaitu dengan cara di modulo dengan nilai i yang terus berubah, apabila hasil modulo adalah 0 maka i adalah faktornya. Jika telah ditemukan hasilnya, maka nilai n akan di update dengan cara dibagi oleh I, dan counter fact akan di increment 1. Line 17 adalah mengupdate nilai ans untuk di outputkan nantinya.

Line 19-21. Jika setelah iterasi loop nilai n masih bersisa lebih dari 1, maka variable ans akan dikalikan dengan nilai 2. Setelah itu di outputkan pada line 21.

HASIL E-OLYMP



Dengan time 2 ms dan memory 72 kb.

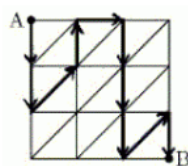
Eolymp 11114. Number of Path

Permasalahan dan pendekatan :

The number of paths

On a checkered sheet of n rows and m columns, find the number of different paths from the upper left corner to the lower right, provided that in one move you can move the checker either down, or up, or to the right, or up - right (diagonally).

You can move a checker to any position no more than once. The figure shows an example of a path on a sheet 3×3 :



Here, the upper left corner is marked with **A** and the lower right corner is marked with **B**. In this example, the number of distinct paths is **343**.

Input

One line contains two integers n and m ($0 \leq n, m \leq 10^9$).

Output

Print the remainder after dividing the number of different paths by **998244353**.



Soal ini pertama meminta 2 nilai interger. N untuk jumlah baris dan m untuk jumlah kolom. Kemudian kita diminta untuk mencari banyaknya cara berbeda yang bisa ditempuh dari titik kiri atas (titik A) kekanan bawah (titik B) dengan pergerakan atas,bawah,kiri,kanan, dan diagonal (kiri bawah ke kanan atas). Contoh testcase untuk 3 x 3 menghasilkan output 343 banyak jalan. Tidak lupa saat mengoutputkan hasil harus di modulo dengan 998244353.

Abstraksi :

Seperti bias akita akan memulai dengan membuat testcase lain untuk melihat pola dan rumus yang ada. Maka akan didapatkan table sebagai berikut:

		m			
		1	2	3	4
n	1	3	9	27	81
	2	5	25	125	625
	3	7	49	343	2401
	4	9	81	729	6561

Dari table bisa dilihat bahwa data yang berada di kolom 1 (yang diberi warna kuning) mengikuti pola $2*n + 1$. Sedangkan total banyaknya Langkah berikutnya bisa didapatkan dari $(2*n + 1)^m$. Oleh karena itu didapat bahwa rumus yang dipakai pada penyelesaian ini adalah

$$(2*n + 1)^m \% 998244353$$

Tetapi jika saat implementasinya menggunakan metode linier akan menghasilkan waktu yang sangat besar. Bayangkan mencoba menghitung perpangkatan hingga 10 pangkat 9. Sehingga diperlukan metode untuk mengurangi kompleksitas perpangkatan menggunakan metode **modular exponential** atau biasa disingkat menjadi modex. Modex adalah cara melakukan perpangkatan dengan mengubah bilangan menjadi berbasis biner.

4.2.4 Modular Exponentiation

In cryptography it is important to be able to find $b^n \bmod m$ efficiently without using an excessive amount of memory, where b , n , and m are large integers. It is impractical to first compute b^n and then find its remainder when divided by m , because b^n can be a huge number and we will need a huge amount of computer memory to store such numbers. Instead, we can avoid time and memory problems by using an algorithm that employs the binary expansion of the exponent n .

Before we present an algorithm for fast modular exponentiation based on the binary expansion of the exponent, first observe that we can avoid using large amount of memory if we compute $b^n \bmod m$ by successively computing $b^k \bmod m$ for $k = 1, 2, \dots, n$ using the fact that $b^{k+1} \bmod m = b(b^k \bmod m) \bmod m$ (by Corollary 2 of Theorem 5 of Section 4.1). (Recall that $1 \leq b < m$.) However, this approach is impractical because it requires $n - 1$ multiplications of integers and n might be huge.

To motivate the fast modular exponentiation algorithm, we illustrate its basic idea. We will explain how to use the binary expansion of n , say $n = (a_{k-1} \dots a_1 a_0)_2$, to compute b^n . First, note that

$$b^n = b^{a_{k-1} \cdot 2^{k-1} + \dots + a_1 \cdot 2 + a_0} = b^{a_{k-1} \cdot 2^{k-1}} \dots b^{a_1 \cdot 2} \cdot b^{a_0}.$$


This shows that to compute b^n , we need only compute the values of $b, b^2, (b^2)^2 = b^4, (b^4)^2 = b^8, \dots, b^{2^k}$. Once we have these values, we multiply the terms b^{2^j} in this list, where $a_j = 1$. (For efficiency and to reduce space requirements, after multiplying by each term, we reduce the result modulo m .)

Kompleksitas waktu yang akan dihasilkan dari teori modex ini adalah $O(\log n)$ tanpa mengurangi tingkat akurasi hasil. Untuk lebih melihat perbedaan waktu yang dihasilkan, kita bisa melihat dari buku Discrete mathematics and Its Application pada permasalahan $3^{644} \bmod 645$, maka langkahnya sebagai berikut :

EXAMPLE 12 Use Algorithm 5 to find $3^{644} \bmod 645$.

Solution: Algorithm 5 initially sets $x = 1$ and $power = 3 \bmod 645 = 3$. In the computation of $3^{644} \bmod 645$, this algorithm determines $3^{2^j} \bmod 645$ for $j = 1, 2, \dots, 9$ by successively squaring and reducing modulo 645. If $a_j = 1$ (where a_j is the bit in the j th position in the binary expansion of 644, which is $(1010000100)_2$), it multiplies the current value of x by $3^{2^j} \bmod 645$ and reduces the result modulo 645. Here are the steps used:

$i = 0$: Because $a_0 = 0$, we have $x = 1$ and $power = 3^2 \bmod 645 = 9 \bmod 645 = 9$;
 $i = 1$: Because $a_1 = 0$, we have $x = 1$ and $power = 9^2 \bmod 645 = 81 \bmod 645 = 81$;
 $i = 2$: Because $a_2 = 1$, we have $x = 1 \cdot 81 \bmod 645 = 81$ and $power = 81^2 \bmod 645 = 6561 \bmod 645 = 111$;
 $i = 3$: Because $a_3 = 0$, we have $x = 81$ and $power = 111^2 \bmod 645 = 12,321 \bmod 645 = 66$;
 $i = 4$: Because $a_4 = 0$, we have $x = 81$ and $power = 66^2 \bmod 645 = 4356 \bmod 645 = 486$;
 $i = 5$: Because $a_5 = 0$, we have $x = 81$ and $power = 486^2 \bmod 645 = 236,196 \bmod 645 = 126$;
 $i = 6$: Because $a_6 = 0$, we have $x = 81$ and $power = 126^2 \bmod 645 = 15,876 \bmod 645 = 396$;
 $i = 7$: Because $a_7 = 1$, we find that $x = (81 \cdot 396) \bmod 645 = 471$ and $power = 396^2 \bmod 645 = 156,816 \bmod 645 = 81$;
 $i = 8$: Because $a_8 = 0$, we have $x = 471$ and $power = 81^2 \bmod 645 = 6561 \bmod 645 = 111$;
 $i = 9$: Because $a_9 = 1$, we find that $x = (471 \cdot 111) \bmod 645 = 36$.

This shows that following the steps of Algorithm 5 produces the result $3^{644} \bmod 645 = 36$. 

Dibanding melakukan looping perkalian sebanyak 644 kali, dengan menggunakan modex hanya cukup melakukan perluangan sebanyak banyak digit dari 644 bila diubah menjadi basis 2 = (1010000100) yaitu 10 kali iterasi. Iterasi tersebut dapat dilakukan dengan **pseudocode** sebagai berikut :

ALGORITHM 5 Fast Modular Exponentiation.

```

procedure modular exponentiation(b: integer,  $n = (a_{k-1}a_{k-2} \dots a_1a_0)_2$ ,
    m: positive integers)
    x := 1
    power := b mod m
    for i := 0 to k - 1
        if  $a_i = 1$  then x := (x · power) mod m
        power := (power · power) mod m
    return x{x equals  $b^n \bmod m$ }

```

Implementasi :

Dengan abstraksi diatas dapat dibuat source code implementasi sebagai berikut dengan parameter fungsi modex menyesuaikan dengan size table.

```

#include<stdio.h>
typedef Long Long ll;

ll modex(ll b, ll e, ll m){
    if (b == 0 || e == 0) return 0;
    ll r = 1;
    while (e > 0){
        if ((e & 1) == 1) r = (r * b) % m;
        e >>= 1;
        b = (b * b) % m;
    }
    return r;
}

int main(){
    ll n, m;
    scanf("%lld %lld", &n, &m);
    printf("%lld\n", modex(n * 2 + 1, m, 998244353));
    return 0;
}

```


Dengan hasil e-olymp

Problem

[The number of paths](#)

Submitted

8 hours ago

Programming Language

C++ 17 (gnu 10.2)

Author

[TimothyHosia_5025211098](#)

100%

C

1 ms

0.07 MiB

Your submission was graded C, which means it passed all tests and used LESS resources than 50% of the submissions on the website.

Test #	Status	Score	Duration	CPU	Memory
✓ Test suite #0	Accepted	100 / 100	1 ms	1 ms	72 KiB
✓ Test #1	Accepted	0 / 0	1 ms	1 ms	72 KiB
✓ Test #2	Accepted	0 / 0	1 ms	1 ms	72 KiB
✓ Test #3	Accepted	20 / 20	1 ms	1 ms	68 KiB
✓ Test #4	Accepted	20 / 20	1 ms	1 ms	72 KiB
✓ Test #5	Accepted	20 / 20	1 ms	1 ms	72 KiB
✓ Test #6	Accepted	20 / 20	1 ms	1 ms	68 KiB
✓ Test #7	Accepted	20 / 20	1 ms	1 ms	68 KiB
		100 / 100	1 ms	1 ms	72 KiB

Dengan waktu 1 ms dan memory 72 kb.