

Solution Tricks For Light Online Judge

বৃহস্পতিবার, ১৩ সেপ্টেম্বর, ২০১২

Solution Of Light Online Judge : 1065 - Number Sequence

In problem 1065 at Light Online Judge, Modular Fibonacci, our task is to calculate huge Fibonacci numbers, modulo some given integer.

INTERPRETATION

We are given two integers, $0 \leq n \leq 2147483647$ and $0 \leq m \leq 20$. We are then asked to calculate and output the n th Fibonacci number, modulo 2^m .

As you can see, n can be pretty big. Let's have a look at a few different methods to compute the n th Fibonacci number.

The naive method, where we do two recursive calls without any optimizations, runs in exponential time. Computing the largest allowed Fibonacci number with this method would take ages, literally.

The dynamic programming method, where we save each computed value (or at least the last two), runs in linear time. We only need a few seconds to compute the largest allowed Fibonacci number using this method, but that's still too slow. We need to be able to run all test cases in under 2 seconds.

There is no specified maximum amount of test cases, but they could be as many as ten, or even hundred, and all of them could possibly be the maximum value.

There is also a [closed-form](#) formula for calculating the n th Fibonacci number, which of course runs in constant time. But it assumes that we are using Real numbers (with infinite precision), which we do not have access to in computers, and thus we cannot use it here.

So what have we learned? A linear-time algorithm is too slow, so we need something faster. A constant-time algorithm would be nice, but seems to be a bit optimistic. We are definitely looking for something there in between, maybe something that takes $\log n$ time?

MATRIX EXPONENTIATION

There is another way to calculate the n th Fibonacci number, that is, using matrix exponentiation.

If you don't know much about matrices, now would be a good time to head over to Khan Academy and take a peek at videos about [linear algebra](#) (specifically "Introduction to matrices" and "Matrix multiplication").

The Fibonacci numbers have the following matrix identity:

$$\begin{pmatrix} \text{fib}_{n+2} & \text{fib}_{n+1} \\ \text{fib}_{n+1} & \text{fib}_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} \text{fib}_{n+1} & \text{fib}_n \\ \text{fib}_n & \text{fib}_{n-1} \end{pmatrix}$$

Now if we put the first Fibonacci numbers into the right-most matrix, and instead of multiplying it once with the other matrix, we multiply it n times, we get:

Or equivalently:

$$\begin{pmatrix} \text{fib}_{n+1} & \text{fib}_n \\ \text{fib}_n & \text{fib}_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

Now we just have to find a fast way to raise a matrix to the n th power.

Let's assume we're trying to raise a number x to the n th power (where $n \in \mathbb{Z}$, meaning that it is an integer). One way would be to multiply x with itself n times. This is pretty slow and needs n multiplications.

A faster algorithm can be derived with the following identity in mind.

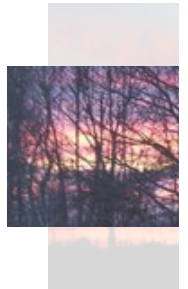
$$x^n = \begin{cases} 1, & \text{if } n = 0 \\ x \cdot x^{n-1}, & \text{if } n \text{ is odd} \\ (x^{\frac{n}{2}})^2, & \text{if } n \text{ is even} \end{cases}$$

The key thing is to notice that we use $x^{n/2}$ twice, but we only need to calculate it once.

Here is a function that calculates x^n using our fast algorithm:

Here is a function that calculates x^n using our fast algorithm:

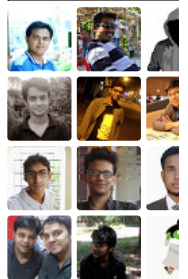
```
1
2
3 public static int pow(int x, int n) {
4     // our base case: x^0 = 1
5     if (n == 0)
6         return 1;
7     // if n is odd
8     if (n % 2 != 0)
9         return x * pow(x, n - 1);
10    // if n is even, calculate x^(n / 2)
11    int sqrt = pow(x, n / 2);
12    // and return it squared
13    return sqrt * sqrt;
14}
```



GOOGLE+ FOLLOW

Sultan Ahmed

চেনাশোনাতে জু:



৯১ আমাকে চেনাশোনা
রাখুন

Sultan Ahmed Sagor | Cre

facebook



Status:
None

সাইন ইন

এই ব্লগটি সন্ধান করুন

জনপ্রিয় পোস্টসমূহ

Hints for Lightoj Begi

15

16

This algorithm, often known as [exponentiation by squaring](#), also works for matrix exponentiation. The only change is that when $n = 0$, we return the [identity matrix](#) instead of 1.

I've included a small matrix library in the complete source code (below). There, the exponentiation method looks like:

```

1
2
3
4 // raise the matrix to the n-th power
5 public Matrix pow(int n) {
6     if (n == 0) {
7         // if n is 0, we return the identity matrix
8         Matrix res = new Matrix(getRows(), getCols());
9         for (int i = 0; i < getRows() && i < getCols(); i++)
10             res.set(i, i, 1);
11         return res;
12     } else if (n % 2 == 0) {
13         // if n is even, return the square root, squared
14         Matrix res = pow(n / 2);
15         return res.multiply(res);
16     } else {
17         // if n is odd, return the matrix multiplied by the matrix to the (n - 1)th power
18         return multiply(pow(n - 1));
19     }
20 }
21
22
23

```

This algorithm only uses about $\log n$ multiplications. Isn't this the running time we were trying to achieve?

IMPLEMENTATION

Now we can calculate the n th Fibonacci number, in a fast way, using matrix exponentiation. The main code now looks like:

```

1
2
3
4 public void run(Scanner in, PrintWriter out) {
5     // initialize the matrix to:
6     // [ 1  1 ]
7     // [ 1  0 ]
8     Matrix fib = new Matrix(2, 2);
9     fib.set(0, 0, 1);
10    fib.set(0, 1, 1);
11    fib.set(1, 0, 1);
12    fib.set(1, 1, 0);
13    // loop through each test case
14    while (in.hasNextInt()) {
15        // read n and m
16        int n = in.nextInt(),
17            m = in.nextInt();
18        // calculate fib^n
19        Matrix fib2 = fib.pow(n);
20        // output the n-th Fibonacci number
21        out.println(fib2.get(1, 0));
22    }
23 }
24
25
26

```

But there is still one thing we have yet to do. The problem asks us to output the n th Fibonacci number, modulo 2^m . We can easily modify our code to do that. We just need to do every computation modulo 2^m . Look inside the complete source for details.

CODE

```

#include<iostream>
#include<cmath>
#include<cstdio>
using namespace std;

class MAT
{

```

Problem

1008 minus a square
several situations just
grid to prompt good.
dark. Attention to the
o...



LightOJ
Down

From
plan the
post the
problem of Light Online
directly. But guys still
help you. ...

Solution Of Light Online 1008 - Fibi's Birthday

```

#include<cstdio>
#include<algorithm>
re(i,a,b) for(int i=a;i<=
#define sf scanf #de
...

```

Solution of Light OJ Problem Makes Problem

Problem Statement :
fond of making easier
discovered a problem
the problem is 'how to
make...

1216 - Juice in the Glass

You have to know
Volume of frustum of
will discuss a little bit
Hope after that you will

LightOJ | Math - Chinese Remainder Theorem Euclid

Change to
plan today, only a quick
Chinese Remainder
the Extended of Euclid



Solution
Online

- Triangles

কোন
বন্ধুরা ?
আমরা এমন একটা সম
করব যাতে আমাদের P
Knowledge (Class 9-
নাগবে। তাহলে চল শুরু

Solution of Light OJ Many Zeroes?

সবাইকে বসন্তের
ভেড়ান ফুলের কলির
আমরা এক নতুন সমস
করতে যাচ্ছি। Problem

Solution of Light OJ Many Points?

Problem Statement :
two points A and B on
plane, output the num
lattice points on the
No...

LightOJ 1014 After Party

ব্লগ সংরক্ষণাগার

► 2013 (9)

▼ 2012 (20)

► November (3)

▼ September (1)

▼ Sep 13 (1)

Solution Of L

- [August](#) (2)
- [June](#) (1)
- [May](#) (6)
- [April](#) (7)

```

public:
    long int a,b,c,d;
}temp,req,cal,base;

int Mod;

MAT& multiply(MAT &A,MAT &B)
{
    temp.a = A.a*B.a+A.b*B.c;
    temp.a=temp.a%Mod;

    temp.b = A.a*B.b+A.b*B.d;
    temp.b=temp.b%Mod;

    temp.c = A.c*B.a+A.d*B.c;
    temp.c=temp.c%Mod;

    temp.d = A.c*B.b+A.d*B.d;
    temp.d=temp.d%Mod;

    return temp;
}

void POW(int n)
{
    if(n==1)
    {
        req = cal;
    }
    else if(n%2)
    {
        POW(n-1);
        req = multiply(cal,req);
    }
    else
    {
        POW(n/2);
        req = multiply(req,req);
    }
}

int main()
{
    int m,i1,test,a1,b1;
    long long int N;
    cin>>test;
    for(i1=1;i1<=test;i1++)
    {
        cin>>a1>>b1>>N>>m;
        Mod=int(pow(10,m));
        cal.a=1,cal.b=1,cal.c=1,cal.d=0;
        base.a=a1+b1,base.b=b1,base.c=b1,base.d=a1;

        if(N==0)
        {
            printf("Case %d: %d\n",i1,a1);
        }
        else if(N==1)
        {
            printf("Case %d: %d\n",i1,b1);
        }
        else if(N==2)
        {
            printf("Case %d: %d\n",i1,a1+b1);
        }
        else
        {
            POW(N-1);
            req = multiply(req,base);
        }
    }
}

```

```
printf("Case %d: %ld\n",i1,req.b);
}
}
return 0;
}
```

We were ask

CONCLUSION

We were asked to calculate huge Fibonacci numbers, and we figured out a fast way to do it with matrix exponentiation. Our algorithm is very fast, calculating the 2147483647th Fibonacci number (modulo 2^{20}) in less than 100 milliseconds. That's pretty good, considering that the 2147483647th Fibonacci number has 448797540 digits.

So what do you think? Did/would you solve it differently? Leave a comment and let me know.

Test Case :

Sample Input

```
6
2 1 11 3
2 1 0 3


0 1 11 3
0 1 42 4
0 1 22 4
0 1 21 4
```

Sample Output

```
Case 1: 199
Case 2: 2
```

```
Case 3: 89
Case 4: 4296
Case 5: 7711
Case 6: 946
```

এর দ্বারা পোস্ট করা [Sultan Ahmed](#) এই সময়ে ৯:২৪ pm

 +3 Google-এ এটির সুপারিশ করুন

লেবেলসমূহ: [1065](#), [Light Online Judge](#), [Number Sequence](#), [Of](#), [Solution](#)

কোন মন্তব্য নেই:

একটি মন্তব্য পোস্ট করুন

আপনার মন্তব্য লিখুন...

এইরূপে মন্তব্য: Unknown (Goc ▼

সাইন আউট

প্রকাশ

পূর্বরূপ

☐ আমাকে সূচিত করুন

[নবীনতর পোস্ট](#)

[হোম](#)

[পুরাতন পোস্ট](#)

এতে সদস্যতা: মন্তব্যগুলি পোস্ট করুন (Atom)



আমার সম্পর্কে



Sultan Ahmed

G+ জন অনুসরণ করছে 91

আমার সম্পূর্ণ প্রোফাইল দেখুন

সদস্য হন

পোস্টগুলি

মন্তব্যসমূহ

সদস্য হন

পোস্টগুলি

মন্তব্যসমূহ

যোগাযোগ ফর্ম

নাম

ইমেল *

বার্তা *

প্রেরণ

সাইন ইন

Picture Window টেমপ্লেট দ্বারা প্রস্তুত Blogger.