Solving Linear Recurrence for Programming Contest

August 20, 2011 · by Ashar Fuadi · 24 Comments

One of the topics that often arises in programming contests nowadays is about solving linear recurrence. It may come as a classic "find the n-th term of Fibonacci sequence" to more complex and creative forms of problems. Usually, after being simplified, the problem is merely asking you the n-th term of a linear recurrence. It should be able to solve with dynamic programming, however, the problem is that usually n can very large.

Because this topic starts to emerge in contests, I would like to share how I usually solve problems of this topic.

Definition

First, let's start with a definition. A linear recurrence relation is a function or a sequence such that each term is a linear combination of previous terms. Each term can be described as a function of the previous terms. A famous example is the Fibonacci sequence: f(i) = f(i-1) + f(i-2). **Linear** means that the previous terms in the definition are only multiplied by a constant (possibly zero) and nothing else. So, this sequence: f(i) = f(i-1) * f(i-2) is **not** a linear recurrence.

Problem

Now, let's state the problem formally.

Given f, a function defined as a linear recurrence relation. Compute f(N). N may be very large.

How to Solve

Here is the method I usually use in programming contests. I break down the method into four steps. Fibonacci sequence will be used as an example.

It is assumed that you have learned matrix and matrix operations, especially multiplication of two matrices.

Step 1. Determine K, the number of terms on which f(i) depends.

More precisely, K is the minimum integer such that f(i) doesn't depend on f(i-M), for all M > K. For Fibonacci sequence, because the relation is:

$$f(i) = f(i-1) + f(i-2),$$

therefore, K = 2. In this way, be careful for missing terms though, for example, this sequence:

$$f(i) = 2f(i-2) + f(i-4)$$

has K = 4, because it can be rewritten explicitly as:

$$f(i) = 0f(i-1) + 2f(i-2) + 0f(i-3) + 1f(i-4).$$

Step 2. Determine vector F_1, the initial values.

If each term of a recurrence relation depends on K previous terms, then it must have the first K terms defined, otherwise the whole sequence is undefined. For Fibonacci sequence (K = 2), the well-known initial values are:

- $f(1) = 1$
- $f(2) = 1$

Although some people may define other initial values, like f(1) = 0 and f(2) = 1, we will use the above definition in this post.

From now on, we will work extensively in matrices. We define a column vector F_i as a K x 1 matrix whose first row is f(i), second row is f(i+1), and so on, until K-th row is f(i+K-1). The initial values of f are given in column vector F_1 that has values f(1) through f(K):

$$F_1 = \begin{bmatrix} f(1) \\ f(2) \\ \vdots \\ f(K) \end{bmatrix}$$

Step 3. Determine T, the transformation matrix.

This is the most important step in solving recurrence relation. The heart of this method is to construct a K x K matrix T, called **transformation matrix**, such that

$$TF_i = F_{i+1}.$$

Here is how to construct it. Suppose that $f(i) = \sum_{j=1}^{K} c_j f(i-j)$. You can solve this equation with any method, and obtain the result:

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ c_K & c_{K-1} & c_{K-2} & c_{K-3} & \cdots & c_1 \end{bmatrix}.$$

More precisely, T is a K x K matrix whose last row is a vector $\begin{bmatrix} c_K & c_{K-1} & \cdots & c_1 \end{bmatrix}$, and for the other rows, the i-th row is a zero vector except that its (i+1)th element is 1.

Let's apply it to our example. In Fibonacci sequence,

$$T = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}.$$

Exercise: find out how matrix T can transforms F_i into F_{i+1}, and memorize the pattern.

Step 4. Determine F_N.

After we construct the transformation matrix, the rest is very simple. We can obtain F_i for any i, by repeatedly multiplying T with F_1. For example, to obtain F_2,

$$F_2 = TF_1.$$

To obtain F_3,

$$F_3 = TF_2 = T^2F_1.$$

And so on. In general,

$$F_N = T^{N-1}F_1.$$

Therefore, the original problem is now (almost) solved: compute F_N as above, and then we can obtain f(N): it is exactly the first row of F_N. In case of our Fibonacci sequence, the N-th term in Fibonacci sequence is the first row of:

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{N-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

What remains is how to compute T^{N-1} efficiently. The most popular method is to use exponentiation by squaring method that works in O(log N) time, with this recurrence:

- $A^p = A$, if $p = 1$,
- $A^p = A.A^{p-1}$, if p is odd,
- $A^p = X^2$, where $X = A^{p/2}$, otherwise.

Multiplying two matrices takes $O(K^3)$ time using standard method, so the overall time complexity to solve a linear recurrence is $O(K^3 \log N)$.

Sample code

Here is a sample C++ code to compute the N-th term of Fibonacci sequence, modulo 1,000,000,007.

```
#include <vector>
#define REP(i,n) for (int i = 1; i <= n; i++)
using namespace std;

typedef long long ll;
typedef vector<vector<ll> > matrix;
const ll MOD = 1000000007;
const int K = 2;

// computes A * B
matrix mul(matrix A, matrix B)
{
    matrix C(K+1, vector<ll>(K+1));
    REP(i, K) REP(j, K) REP(k, K)
        C[i][j] = (C[i][j] + A[i][k] * B[k][j]) % MOD;
    return C;
}

// computes A ^ p
matrix pow(matrix A, int p)
{
    if (p == 1)
        return A;
    if (p % 2)
        return mul(A, pow(A, p-1));
    matrix X = pow(A, p/2);
    return mul(X, X);
}

// returns the N-th term of Fibonacci sequence
```

```
31 int fib(int N)
32 {
33     // create vector F1
34     vector<ll> F1(K+1);
35     F1[1] = 1;
36     F1[2] = 1;
37
38     // create matrix T
39     matrix T(K+1, vector<ll>(K+1));
40     T[1][1] = 0, T[1][2] = 1;
41     T[2][1] = 1, T[2][2] = 1;
42
43     // raise T to the (N-1)th power
44     if (N == 1)
45         return 1;
46     T = pow(T, N-1);
47
48     // the answer is the first row of T . F1
49     ll res = 0;
50     REP(i, K)
51         res = (res + T[1][i] * F1[i]) % MOD;
52     return res;
53 }
```

Variants

Those are the basics of solving linear recurrence. Here we will discuss the variants that often occur.

*** Constants in the relation**

The recurrence relation may include a constant, i.e., the function is of the form $f(i) = \sum_{j=1}^{K} c_j f(i-j) + d$. In this variant, the vector F is enhanced to "remember" the value of d. It is of size (K+1) x 1 now:

$$F_i = \begin{bmatrix} f(i) \\ f(i+1) \\ \vdots \\ f(i+K-1) \\ d \end{bmatrix}.$$

We now want to construct a matrix T, of size (K+1) x (K+1), such that:

$$[T] \begin{bmatrix} f(i) \\ f(i+1) \\ \vdots \\ f(i+K-1) \\ d \end{bmatrix} = \begin{bmatrix} f(i+1) \\ f(i+2) \\ \vdots \\ f(i+K) \\ d \end{bmatrix}.$$

After learning the basic method, can you figure out the matrix T for this variant? Here it is:

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ c_K & c_{K-1} & c_{K-2} & c_{K-3} & \cdots & c_1 & 1 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}.$$

For example, let $f(i) = 2f(i-1) + 3f(i-2) + 5$. Then, the matrix equation for this function is:

$$\begin{bmatrix} 0 & 1 & 0 \\ 3 & 2 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f(i) \\ f(i+1) \\ 5 \end{bmatrix} = \begin{bmatrix} f(i+1) \\ f(i+2) \\ 5 \end{bmatrix}.$$

*** Odd/even conditional function**

The function f may behave differently according to the parity of the argument. For example, if i is even, then $f(i) = f(i-1)/2$, otherwise (if i is odd) $f(i) = 3f(i-1) + 1$.

How to solve this variant? It is possible to split the equation into even and odd parts. We construct two matrices T_{even} and T_{odd} such that:

- $T_{\text{even}} F_i = F_{i+1}$, i is even
- $T_{\text{odd}} F_i = F_{i+1}$, i is odd

We also construct matrix $T = T_{\text{even}}.T_{\text{odd}}$. Now, we can compute F_N with a single formula:

- $F_N = T^{N/2}.F_1$, if N is odd,
- $F_N = T_{\text{odd}}.T^{(N-1)/2}.F_1$, otherwise.

Note that T_{even} and T_{odd} must be of the same size, so in this example, convert $f(i) = f(i-1)/2$ into $f(i) = \frac{1}{2}f(i-1) + 0$ to make it look like the odd part, and then apply the standard method.

*** Your variants**

Have you ever encountered another variant of linear recurrence problem? Please share it in the comment section 😃

Related posts:

- Solving Linear Recurrence for Programming Contest – Examples
- 3 Ways to Define Comparison Functions in C++
- Polyglot: Write Once, Compile Anywhere
- CodeSprint 2 Interview Street Problems Analysis

filed under: tips · tagged: c++, linear recurrence

About Ashar Fuadi
Ashar Fuadi is a competitive programmer from University of Indonesia. He loves to code, especially for TopCoder SRM, Codeforces, and ICPC.
Follow Ashar on Google+ and Twitter.

Comments

Nadeem Moidu says:
August 20, 2011 at 10:40

One variation you can easily handle with this strategy is recurrence containing more than one functions.
e.g.

f(n) = f(n-1) + g(n-1)
g(n) = f(n-1) + g(n-1)
I think it is possible to convert these kind of recurrences to a single function recurrence, but when you are using the matrix exponentiation, it is easier to use it straight away.

fushar says:
August 21, 2011 at 00:43

“but when you are using the matrix exponentiation, it is easier to use it straight away”

What is the “straight away” method?

Nadeem Moidu says:
August 21, 2011 at 02:21

You’re Fi will now look as follows
Fi =
[f(i) f(i+1) g(i) g(i+1)]

Now isn’t it easy to come up with the transformation matrix?

Anil says:
June 18, 2012 at 16:20

Could you please post the transformation matrix in the Odd and Even Variants case. I need to verify once .

fushar says:
June 19, 2012 at 10:03

Hello Anil,

T_even:
0.5 0
0 1

T_odd:
3 1
0 1

elaine says:
June 22, 2012 at 12:03

some example of recurrences codes that have to count on its own like for example on this equation

5!= 120
3!=6

could u give me some tips to solve this program ?

thanks alot

Anil says:
June 22, 2012 at 15:23

" We also construct matrix T = T_{\text{even}} . T_{\text{odd}}. Now, we can compute FN with a single formula:

F_{N} = T^{N/2} . F_{1} \text{, if } N \text { is odd,}
F_{N} = T_{\text{odd}} . T^{(N-1)/2} . F_{1} \text{, otherwise.} "

In the above quote , I guess there is some discrepancy. Are you sure that it is true ? I guess the conditions are exchanged.

Anil says:
June 22, 2012 at 15:28

Sorry . I got the point . It is completely correct .

coder says:
July 9, 2012 at 05:28

I think it is:

F_{N} = T^{(N-1)/2} . F_{1} \text{, if } N \text { is odd,}
F_{N} = T_{\text{even}} . T^{(N-2)/2} . F_{1} \text{, otherwise.}

Correct me if it is wrong.

Thanks.

fushar says:
July 9, 2012 at 10:21

I think mine is correct. Note that by '/' I meant an integer division 😃

Gaurav says:
August 11, 2012 at 00:50

What in case of f(n)=f(n-1)+f(n-2) +/- n??

Unknown says:
September 2, 2012 at 23:45

Thank you for this nice post .
How to handle recurence involving two separate functions:
For eg->
f[n]=f[n-1]+f[n-2]+g[n] where g[n]=g[n-1]+g[n-2] ??

Boxer says:
September 9, 2012 at 06:24

Solution :
g(n) = f(n) – f(n-1) – f(n-2) => g(n-1) = f(n-1) – f(n-2) – f(n-3) …..(I)
and g(n-2) = f(n-2) – f(n-3) – f(n-4)……(II)

of (I)+(II) g(n-1) + g(n-2) = f(n-1) – f(n-2) – f(n-3) + f(n-2) – f(n-3) – f(n-4)…(III)
as g(n) = g(n-1)+g(n-2) => g(n) = f(n-1) – f(n-2) – f(n-3) + f(n-2) – f(n-3) – f(n-4)….(IV)
then g(n) = f(n-1) – f(n-4)
finally f(n) = f(n-1) + f(n-2) + f(n-1)-f(n-4) = 2f(n-1)+f(n-2)-f(n-4)

Igor says:
February 21, 2013 at 22:45

The article is great!
There is a small bug in the code:
either MOD = 1000000007 instead of MOD = 10000000007
or res = (res + (T[1][i] * F1[i]) % MOD) % MOD instead of res = (res + T[1][i] * F1[i]) % MOD;
Best regards

Ashar Fuadi says:
March 3, 2013 at 11:13

@ Igor: Thanks, this is fixed.

Abhishek Panigrahy says:
March 20, 2013 at 11:52

How can i solve a recurrence relation of the form :
F(n) = F(n-3) + n/2 + 1

Nikhil Sharma says:
March 27, 2013 at 17:58

How to solve the recurrence of the following specific type

T(n+1)=T(n)+ 26^(n+1)/2

Florin says:
June 18, 2013 at 18:45

Thank you for this great post!

unknown says:
July 16, 2013 at 00:18

Can anyone tell me how the matrix turns out to be {2,2,1,0} for f(n) = f(n-1)+4f(n-2)+2f(n-3) in the below link?
http://www.codechef.com/viewsolution/547362

Kaidul says:
September 26, 2013 at 04:14

Can you please tell me the linear recurrence equation for
http://mukeshiiitm.wordpress.com/2011/06/25/spoj-7487-flibonakki/ ?

Michael says:
December 22, 2013 at 00:33
That was really helpful

Mostafa says:
January 20, 2014 at 14:44
This has been the clearest explanation I have found on the topic among the blogs I have looked at. Thanks a lot :D. Also, can you recommend any books that go more in-depth on the topic?

Daniel Flam says:
August 22, 2014 at 03:27
if Ck is 1 (such as in the case of fibonacci series) then you can lose F1 in the math and the
actual result is for all n>2 T^(n-1)[1,1] is the nth series element. saves a matrix multiplication.

BigBash says:
September 7, 2014 at 23:34
How can we solve recurrence relations involving square roots like F(N)=F(N-1)+F(N-2)+sqrt(3*F(N-2)+F(N-1)) using matrix exponentiation ?

Speak Your Mind

Name *

Email *

Website

Post Comment