demoralizer's blog

[Tutorial] Solving Linear Recurrences with various methods, Including O(N logN logK) using FFT

By demoralizer, history, 17 months ago,

Hi this is my first educational blog on codeforces......

I have been procrastinating to write this one for over 15 months by now, but thanks to **crackersamdjam**'s comment, I finally did it!

Any suggestions or improvements or constructive-criticism to the blog, is heavily appreciated.

In this blog, I tried to avoid too many technical terms (especially in the last section), and tried to make it beginner friendly.

Linear Recurrence : Introduction

Fibonacci Sequence is one of the simplest examples of a linear recurrence:
$F_x = F_{x-1} + F_{x-2}$, with $F_0 = 0, F_1 = 1$

Here is a more general example of a linear recurrence:

$$R_x = \sum_{i=1}^{n} C_i \cdot R_{x-i}$$

where C_i is constant and R_x is the x-th term of the recurrence. Since R_x depends on the previous n terms of the recurrence, it is called a linear recurrence of order n. It is called a "Linear" Recurrence, since we do not have any terms of the type $R_x \cdot R_y$

Note: We need n initial conitions for a linear recurrence of order n, for example, we have 2 initial conditions for the Fibonacci Sequence.

Iternary

In this blog we will learn about various methods of solving the following problem: Given a Linear Recurrence of order N, find the K-th term of the recurrence. There are various methods to do this:

- $O(N \cdot K)$ using DP
- $O(N^3 \cdot logK)$ using Matrix Exponentiation
- $O(P(n) \cdot logK)$ using Characterstic Polynomials where $P(n)$ is the time required for polynomial multiplication — which can be $O(N^2)$ naively or $O(N^{1.58})$ using Karatsuba Multiplication or $O(NlogN)$ using Fast Fourier Transform.

DP Solution

The $O(N \cdot K)$ solution is pretty trivial. (Assume `dp[0]` .. `dp[n-1]` are stored correctly as initial conditions)

```
for(int i = n; i < k; i++){
    dp[i] = 0;
    for(int j = 1; j <= n; j++){
        dp[i] += c[j] * dp[i - j];
    }
}
```

Matrix Exponentiation

You can find more detailed explanations on this technique here and here. But here I've described it briefly:

Let's look at the problem from another angle. From the DP solution, it is clear that we need to keep track of the previous n elements at all times, and it won't matter even if we "forget" other elements. So let's keep a column vector of the "current" n consecutive elements. Since the recurrence relation is a Linear Relation, it is possible to have a linear transformation to get the next elements.

$$\begin{bmatrix} R_{n-1} \\ R_{n-2} \\ \vdots \\ R_0 \end{bmatrix} \xrightarrow{\text{A Linear Transformation}} \begin{bmatrix} R_n \\ R_{n-1} \\ \vdots \\ R_1 \end{bmatrix}$$

Finding this linear transformation is quite intuitive and it can be expressed as multiplication with a square matrix, so let's directly write the general result.

$$\begin{bmatrix} C_1 & C_2 & \cdots & C_{n-1} & C_n \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} R_{i+(n-1)} \\ R_{i+(n-2)} \\ R_{i+(n-3)} \\ \vdots \\ R_{i+(0)} \end{bmatrix} \implies \begin{bmatrix} R_{i+(n)} \\ R_{i+(n-1)} \\ R_{i+(n-2)} \\ \vdots \\ R_{i+(1)} \end{bmatrix}$$

Reader is advised to take a moment to manually mutliply the matrices and verify the above result for a better understanding.

Alright, so a multiplication with the Transformation Matrix shifts the elements of the vector by 1 index each, so we can shift it by X indices by multiplying the transformation matrix to it X times. However we take advantage of the fact that we can first multiply the transformation with itself X times and then multiply it with the column vector. Why is this advantageous? Because we can use Binary Exponentiation! If we the transformation matrix is T, we can find T^X in $O(N^3 logX)$ time. This lets us get the K-th term in $O(N^3 logK)$ time.

Check out this Mashup for practice problems

Using Characterstic Polynomial

I might've over-explained this section of the blog, because I personally found it very hard to understand this part when I initally learnt it.

I learnt about this technique from ***TLE****'s blog on Berlekamp Massey Algorithm, but since it wasn't explained in detail, I had a lot of difficulty in understanding this, so I'll try to explain it in a more beginner-friendly way. Apparently the name of this trick is Kitamasa Method.*

First of all, an important observation

The original recurrence can be re-written as:

$$R_j - \sum_{i=1}^{n} C_i \cdot R_{j-i} = 0$$

Which means, any multiple of $(R_j - \sum_{i=1}^{n} C_i \cdot R_{j-i})$ is 0 and if it is added or subtracted anywhere, the answer doesn't change.

Let's bring in Polynomials

In simple words, in the linear recurrence, replace R_i with x^i everywhere to a get a polynomial equation. Multiplying x to every element of the polynomial would just mean adding 1 to all the subscripts in the linear recurrence. It is quite obvious to see that adding or subtracting such polynomial equations is also allowed. *"Allowed? For what?"* — **For converting the polynomial back to the recurrence!** Yes! This is a bit magical.

Try to check it on a few cases manually to verify that it is always allowed to do the following:

- Convert the linear recurrence into polynomial equation
- Add/Subtract/Multiply the polynomial equation with x or some constants
- Convert the polynomial back to linear recurrence (replace x^i with R_i everywhere)

The final equation that you get after this, will be valid.

Alright, so what now? Let's convert the original recurrence into a polynomial (It is called the **Characterstic Polynomial** of the Reccurence)

$$f(x) = x^n - \sum_{i=1}^{n} C_i \cdot x^{n-i}$$

Now since we know that upon converting the characterstic polynomial back to recurrence form, its value will be 0. We take advantage of this — Any multiple of the characterstic polynomial can be added or subtracted without any effect on the result.

We want to find R_K, upon converting it to polynomial we get x^K

We wish to find $\left(x^K + m(x) \cdot f(x)\right)$, where $m(x)$ is any polynomial! (Since $f(x)$ will be 0 upon converting back to recurrence)

Now the idea is simple, we choose $m(x)$ in such a way that, only terms with degree less than n remain. To do so, we pick, $m(x)$ as the negative of the quotient when x^K is divided by $f(x)$. Which eventually leaves us with $x^K \ mod \ f(x)$

Hence the final solution

Find $x^K \ mod \ f(x)$ and then convert it to recurrence, which will have only terms $R_0, R_1, \cdots, R_{n-1}$

They way to do this is:

- Figure out how to calculate poly mod poly. (remainder when a polynomial is divided by another polynomial)
- Use binary exponentiation to find $x^K \ mod \ f(x)$

It is easy to see that there will be $O(logK)$ steps, with each step involving poly mod poly.

Hence using FFT, this can be done in $O(NlogNlogK)$

"Wait! You didn't tell how to do Poly mod Poly!"

Now, I won't be explaining how to do Poly mod Poly, because it is pretty easy to do naively in $O(N^2)$, and very complex using FFT in $O(NlogN)$ — You can read about it here.

However, you can just use Poly mod Poly as a **black box** and use this code library by **adamant** (Although it is not very optimized, but it is good for basic use)

Practice Problems

RNG Codechef

You can also try the problems from the matrix exponentiation mashup using this technique, but it'll be an overkill.

+299 demoralizer 17 months ago 10

Comments (10)

Write comment?

vishaaaal

17 months ago, # | ← Rev. 2 +43

Thanks for this tutorial!

Also, another great tutorial, which I used to learn about Berlekamp-Massey.

A problem which appeared in an Edu Round.
Also one more practice problem and editorial.

→ Reply

Golovanov399

17 months ago, # | ← Rev. 2 +48

Given the characteristic polynomial, one can calculate the K-th term faster than you suggested, that is, without poly mod poly (but with the same time complexity). See my comment from the other blog.

Also, here one can check their implementation

→ Reply

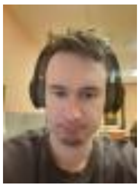
17 months ago, # | ☆ ← Rev. 5 +34

Hmm, I was wondering if there is any meaningful way to explain the $x^k \bmod f(x)$ part with as little direct manipulations with coefficients as possible. I know that we may just directly show it by comparing rows of C^{n+1} and C^n matrices, but it's somewhat ugly to me. And the approach in this blog doesn't seem rigorous enough. What if we look on it from generating functions perspective?

For $g(x) = \sum\limits_{i=0}^{\infty} R_i x^i$, the linear recurrence essentially means that

$$g(x)t(x) = g(x)t(x) \bmod x^n,$$

where $t(x) = 1 - \sum\limits_{i=1}^{n} C_i x^i$. The modulo part here means that $g(x)t(x)$ has zero coefficient ($\iff$ the recurrence stands) near x^k for every $k \geq n$ and for $k < n$ the coefficient is not restricted in any way. Thus, $g(x)$ may be represented as



adamant

$$g(x) = \frac{a(x)}{t(x)}$$

where $a(x) = g(x)t(x) \bmod x^n$. Since the product is taken modulo x^n, only first n coefficients of $g(x)$ matter for the definition of $a(x)$. Note that $t(x) = x^n f(x^{-1})$ where $f(x)$ is the characteristic function defined in this blog. Because of that, $d(x) = g(x^{-1})$ might be represented as

$$d(x) = \frac{a(x^{-1})}{t(x^{-1})} = \frac{x^n a(x^{-1})}{f(x)}$$

Noteworthy, $x^k d(x)$ has R_k as the coefficient near x^0. On the other hand, $x^t f(x) d(x) = x^{n+t} a(x^{-1})$ always has zero coefficient near x^0 for $k \geq 0$, as $a(x)$ is the polynomial of degree at most $n-1$.

Now, to find the linear combination of $R_0, \dots, R_{n-1}$ that yields R_k is the same as to find a polynomial $p(x)$ of degree at most n such that coefficients near x^0 in $x^k d(x)$ and $p(x)d(x)$ are the same.

But since for $x^t f(x) d(x)$ this coefficient is 0, we may conclude that polynomials $p(x)$ and $p(x) \bmod f(x)$ yield the same coefficient near x^0 and, thus, it is indeed safe to go with $p(x) = x^k \bmod f(x)$.

→ Reply

17 months ago, # ^ | ☆ +23

demoralizer

Thanks for this Mathematical Proof of the technique!

Before this, I always thought of this technique as a fast way of substituting the value of the highest order term repeatedly, and assumed that the operations involved are just "coincidentally" the same as poly mod poly.

→ Reply

16 months ago, # ^ | ☆ ← Rev. 3 +5

islingr

This digression seems rather unecessary to me, the idea given in TLE's blog works just fine.

For the recurrence $a_k = \sum_{i=1}^n c_i a_{k-i}$. Define the linear functional $T : \mathbb{F}[x] \to \mathbb{F}$ as $T(x^k) = a_k$.

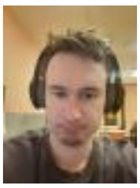
Now, if f is the characteristic polynomial defined in the blog, $T(x^r f) = 0$ for any $r \geq 0$ by definition of the recurrence. That is, $\operatorname*{span}_{r\geq 0}(x^r f) \subseteq \ker T$. This span is nothing but the polynomial multiples of f, call it $f\mathbb{F}[X]$.

You're done here, you see that $T(p) = T(q)$ for any $p - q \in f\mathbb{F}[x]$ so $T(x^k) = T(x^k \mod f)$.

If you wish, you can view it as a linear functional over $\mathbb{F}[x]/f\mathbb{F}[x]$ but that's just adding more needless jargon. Really all we did here was write the "direct manipulations" here in a linear algebra setting.

→ Reply

16 months ago, # ^ | ☆ ← Rev. 2 0



adamant

Yeah, that's nice way to see it. As I understand, $T(p) = [x^0]p(x)g(x^{-1})$ in my terms... So, I essentially do the same stuff, but a bit overly verbose, perhaps.

→ Reply

17 months ago, # | ☆ ← Rev. 6 +19

oToToT

The first problem I tried with this technique is this atcoder problem by **E869120**. (The whole series of problems are good for beginners.)

The problems goes like:

Given N, K, find the number (mod 998244353) of possible non-negative sequence A_i with size N satisfying $\forall 1 \leq l \leq r \leq N$,

$\min(A_l, A_{l+1}, \ldots, A_r) \times (r - l + 1) \geq K$.

→ Reply

17 months ago, # | ☆ -48

MarkZuckerberg

demoralizer finally did it.

I WAS THE FIRST PERSON WHO REQUESTED IT !

https://codeforces.com/blog/entry/83164?#comment-704678

→ Reply

17 months ago, # | ☆ -37

dthompson

Are you sure Berlekamp Massey isn't better? There are a lot of log factors in your code too.

→ Reply

17 months ago, # ^ | ☆ +16

nor

- Berlekamp Massey has a different purpose: "interpolate" a linear recurrence from the first few values, while the blog concerns itself with just computing some element of a given linear recurrence.
- Why is having log factors bad? N^{100} is worse than $N \log^{100} N$, isn't it?

→ Reply