



## -is-this-fft-'s blog

### [Tutorial] FFT

 By [-is-this-fft-](#), [history](#), 3 years ago, 

If I was a YouTuber, this would be the place where I fill the entire screen with screenshots of people asking me to make this. Anyway, not so long ago I gave a lecture on FFT and now [peltorator](#) is giving away [free money](#), so let's bring this meme to completion.

In this blog, I am going to cover the basic theory and (competitive programming related) applications of FFT. There is a long list of generalizations and weird applications and implementation details that make it faster. Maybe at some point I'll write about them. For now, let's stick to the basics.

**The problem.** Given two arrays  $a$  and  $b$ , both of length  $n$ . You want to quickly and efficiently calculate another array  $c$  (of length  $2n - 1$ ), defined by the following formula.

$$c_k = \sum_{i+j=k} a_i \cdot b_j.$$

Solve it in  $O(n \log n)$ .

First of all, why should you care? Because this kind of expression comes up in combinatorics all the time. Let's say you want to count something. It's not uncommon to see a problem where you can just write down the formula for it and notice that it is exactly in this form.

FFT is what I like to call a very black-boxable algorithm. That is, in roughly 95% of the FFT problems I have solved, the **only thing you need to know about FFT** is that FFT is the key component in an algorithm that solves the problem above. You just need to know that it exists, have some experience to recognize that and then rip someone else's super-fast library off of [judge.yosupo.jp](#).

But if you are still curious to learn how it works, keep reading...

### The strategy

By the way, the array  $C$  we are calculating is called a *convolution* of  $A$  and  $B$ . There are other kinds of convolutions, but this is the kind we care about here. Anyway, does the expression for  $C$  remind you of anything?

Polynomials! Imagine that the arrays  $A$  and  $B$  were the coefficients of some polynomials.

$$\begin{aligned} A(z) &= a_0 z^0 + a_1 z^1 + \dots + a_{n-1} z^{n-1}, \\ B(z) &= b_0 z^0 + b_1 z^1 + \dots + b_{n-1} z^{n-1}. \end{aligned}$$

If we look at things this way, what would  $c$  represent? It turns out that  $c$  is *exactly* the coefficients of the polynomial  $C(z) = A(z) \cdot B(z)$ . If you don't see why, I recommend you write the multiplication out with pen and paper. By the way, I am denoting the variable with  $z$  here because we are going to work with complex numbers, and it is customary to write complex variables as  $z$  instead of  $x$ .

[Crash course on complex numbers](#)

So, we now know that we want to multiply two polynomials. But how does that help us? For one, imagine if we, instead of the *coefficients* of  $A$  and  $B$ , knew the *values* of  $A$  and  $B$  for a number of fixed  $z$ . That is, we have decided on some numbers  $z_0, z_1, z_2, \dots, z_{m-1}$  and know all of the following:

$$\begin{aligned} &A(z_0), A(z_1), A(z_2), \dots, A(z_{m-1}) \\ &B(z_0), B(z_1), B(z_2), \dots, B(z_{m-1}) \end{aligned}$$

#### → Pay attention

**Before contest**  
[Codeforces Global Round 30 \(Div. 1 + Div. 2\)](#)  
 42:16:35  
[Register now »](#)  
 \*has extra registration

#### → pastilia

Rating: **1575**  
 Contribution: 0



pastilia

- [Settings](#)
- [Blog](#)
- [Teams](#)
- [Submissions](#)
- [Problemsetting](#)
- [Groups](#)
- [Talks](#)
- [Contests](#)

#### → Top rated

#	User	Rating
1	<a href="#">tourist</a>	3747
1	<a href="#">Benq</a>	3747
3	<a href="#">jiangly</a>	3705
4	<a href="#">orzdevinwang</a>	3670
5	<a href="#">ksun48</a>	3653
6	<a href="#">Kevin114514</a>	3651
7	<a href="#">ecnerwala</a>	3555
8	<a href="#">Radewoosh</a>	3554
9	<a href="#">Nachia</a>	3549
10	<a href="#">Um_nik</a>	3427

[Countries](#) | [Cities](#) | [Organizations](#)
[View all →](#)

#### → Top contributors

#	User	Contrib.
1	<a href="#">Um_nik</a>	168
2	<a href="#">Qingyu</a>	166
3	<a href="#">errorgorn</a>	165
4	<a href="#">adamant</a>	155
5	<a href="#">cry</a>	152
5	<a href="#">Dominator069</a>	152
7	<a href="#">Proof_by_QED</a>	149
8	<a href="#">soulless</a>	144
9	<a href="#">awoo</a>	141
10	<a href="#">SecondThread</a>	140

[View all →](#)

#### → Find user

Handle:

Find



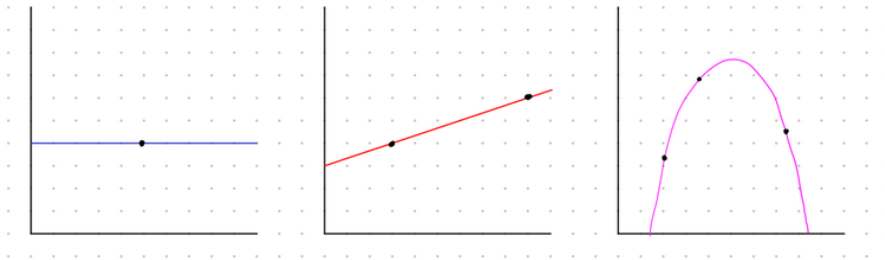
Finding the same values for  $C$  is now dead simple. We just calculate  $C(z_0)$  as  $A(z_0) \cdot B(z_0)$ ,  $C(z_1)$  as  $A(z_1) \cdot B(z_1)$  and so on.

Also, knowing the values of a polynomial is not worse than knowing its coefficients. Knowing the value of a polynomial  $C(z)$  for sufficiently many different  $z$  is enough information to find out what the coefficients of  $C$  are.

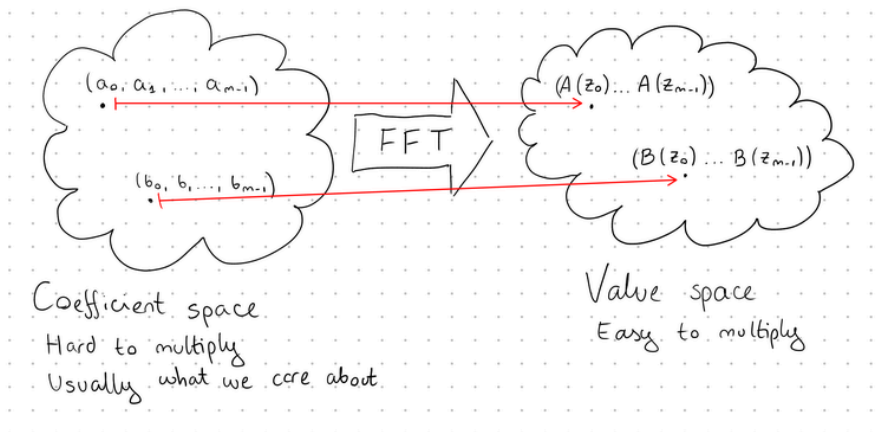
**Fact.** Given  $m$  pairs  $(z_0, w_0), (z_1, w_1), \dots, (z_{m-1}, w_{m-1})$  where all the  $z$ -s are distinct, there is exactly one polynomial  $P$  with degree up to  $m - 1$  such that  $P(z_i) = w_i$  for each  $i$ .

For example:

- given a point on a plane, there is exactly one way to draw a horizontal line (i.e. a constant polynomial) through it.
- given two points on a plane with different  $x$ -coordinates, there is exactly one way to draw a line (i.e. linear polynomial) through them.
- given three points on a plane with different  $x$ -coordinates, there is exactly one way to draw a parabola (i.e. quadratic polynomial) through them.



If you know the *coefficients* of polynomials, it is easy to *add* them, but hard to *multiply* them. To efficiently multiply polynomials, we need to know the *values* of polynomials. However, the coefficients of polynomials are what we are usually (maybe indirectly) given, and often also what we need to work with after multiplying. Therefore, we need a bridge between them. An algorithm that can quickly transport you from coefficient-world to value-world. That algorithm is the fast Fourier transform.



It should be noted that FFT doesn't calculate the values of the polynomial for arbitrary  $z_0, \dots, z_{m-1}$ , but rather one specific set.

## The core of the algorithm

Now we are ready to tackle the hard part. We're given a polynomial  $A(z) = a_0 z^0 + \dots + a_{m-1} z^{m-1}$ . First, let's make life easier for us and assume  $m$  is always a power of two, say  $2^k$ . It can't hurt and we can easily do so by appending zeros to the end of the array of coefficients. Anyway, we need to calculate the value of  $A(z)$  for  $n$  distinct  $z$ -s.

Which  $z$ -s? We haven't decided yet, and we only need one set. It is pretty clear that we need to make the choice carefully, and that the algorithm will need to exploit some special properties of these values. But the crash course above hints at the correct choice, so I'm just going to make it. Hopefully it will become clear why. I define

$$z_j = \exp\left(\frac{2\pi i j}{m}\right).$$

## → Recent actions

MateyKnows → [Help](#)

iori\_yagggggami → [difference between 1100 and 1200?](#)

Lever → [My obsession](#)

aryanv → [Iranian Combinatorics Olympiad 2025](#)

Proof\_by\_QED → [Testing Round 20 \[Communication Problems\]](#)

-is-this-fft- → [Run-twice problems with Polygon and testlib.h](#)

Sul\_A. → [Should CF rounds have communication problems?](#)

mxdycn → [Xor problem](#)

IceBorworntat → [Behind the Scenes: Chairing the ICPC Regional Bangkok 2025 Technical Committee](#)

PineapplesOnPizza → [The 2025 ICPC Asia Bangkok Regional Contest \[Online Mirror\]](#)

Kunal\_Dev → [I Am Curious... How Do Problem Setters Come Up With New Problems?](#)

DharunKaarthick → [Importance of logging out part 4](#)

Dynamic\_Immortal → [Importance of logging out part 3.](#)

cry → [Codeforces Round 971 \(Div. 4\) Editorial](#)

Sul\_A. → [Testing Round 20 Editorial \(officially unofficial\)](#)

Noobish\_Monk → [Suggestion for communication problems](#)

E869120 → [I should have cared about personal computer](#)

Proof\_by\_QED → [Codeforces Round 1047 \(Div. 3\) Editorial](#)

Edvard → [Editorial of Educational Codeforces Round 2](#)

kostka → [IOI 2026 Call for Tasks](#)

MathModel → [Invitation to Eolymp Weekend Practice #13](#)

MidnightCodeCup → [See you at Midnight?](#)

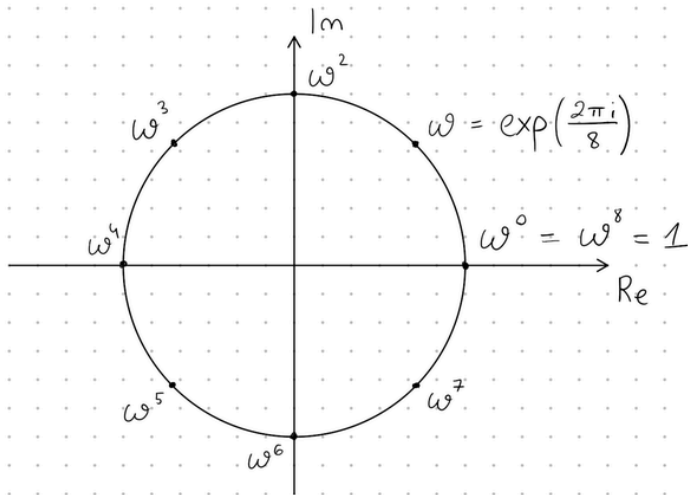
Endagorion → [Pinely Round 5 editorial](#)

gyh20 → [Codeforces Round #670 \(Div. 2\) Editorial](#)

Rising-coder → [Complaining Again about Round 1056!](#)

[Detailed →](#)

The numbers  $z_0, z_1, z_2, \dots, z_{m-1}$  go along the unit circle of the complex plane, evenly spaced apart. It gets better. If I define  $\omega = z_1 = \exp\left(\frac{2\pi i}{m}\right)$ , then  $z_j = \omega^j$ . Convenient. These numbers have a neat property:  $\omega^0 = \omega^m = 1$ ,  $\omega^1 = \omega^{m+1}$  etc. The exponents behave as though we are adding them, modulo  $m$ .  $\omega$  is an omega, not a double-u.



**Definition.** Given an array  $a_0, a_1, \dots, a_{m-1}$ , the *fast Fourier transform* is any algorithm that calculates the tuple  $(A(\omega^0), A(\omega^1), \dots, A(\omega^{m-1}))$  in  $O(m \log m)$  time. The transform itself is called the *discrete Fourier transform*. We denote

$$\text{DFT}(a_0, a_1, \dots, a_{m-1}) = (A(\omega^0), A(\omega^1), \dots, A(\omega^{m-1})).$$

Now let's explain one possible FFT algorithm. This is the most well-known one, also known as the *Cooley-Tukey algorithm*. It is convenient to explain the algorithm by doing basic matrix operations, so let's translate the problem to matrix form. Given  $a_0, a_1, \dots, a_{m-1}$ , we need to calculate  $A(\omega^0), A(\omega^1), \dots, A(\omega^{m-1})$ . In matrix form, this means calculating

$$\begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \dots & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \dots & \omega^{m-1} \\ \omega^0 & \omega^2 & \omega^4 & \dots & \omega^{2(m-1)} \\ \omega^0 & \omega^3 & \omega^6 & \dots & \omega^{3(m-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \omega^0 & \omega^{m-1} & \omega^{2(m-1)} & \dots & \omega^{(m-1)(m-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{m-1} \end{bmatrix}.$$

Let's do an example with  $m = 8$ . This will hopefully be sufficient to explain the general idea. We need to calculate

$$\begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 & \omega^0 & \omega^2 & \omega^4 & \omega^6 \\ \omega^0 & \omega^3 & \omega^6 & \omega^1 & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ \omega^0 & \omega^4 & \omega^0 & \omega^4 & \omega^0 & \omega^4 & \omega^0 & \omega^4 \\ \omega^0 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega^1 & \omega^6 & \omega^3 \\ \omega^0 & \omega^6 & \omega^4 & \omega^2 & \omega^0 & \omega^6 & \omega^4 & \omega^2 \\ \omega^0 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega^1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix}.$$

The exponents in this matrix are the multiplication table, taken modulo 8 because  $\omega^0 = \omega^8$ .

Let's reorder the columns of this matrix by bringing the even-numbered columns to the left and the odd-numbered columns to the right. Note that this will also reorder the rows of the vector.

$$\begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 & \omega^1 & \omega^3 & \omega^5 & \omega^7 \\ \omega^0 & \omega^4 & \omega^0 & \omega^4 & \omega^2 & \omega^6 & \omega^2 & \omega^6 \\ \omega^0 & \omega^6 & \omega^4 & \omega^2 & \omega^3 & \omega^1 & \omega^7 & \omega^5 \\ \hline \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^4 & \omega^4 & \omega^4 & \omega^4 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 & \omega^5 & \omega^7 & \omega^1 & \omega^3 \\ \omega^0 & \omega^4 & \omega^0 & \omega^4 & \omega^6 & \omega^2 & \omega^6 & \omega^2 \\ \omega^0 & \omega^6 & \omega^4 & \omega^2 & \omega^7 & \omega^5 & \omega^3 & \omega^1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_2 \\ a_4 \\ a_6 \\ a_1 \\ a_3 \\ a_5 \\ a_7 \end{bmatrix}$$

The quadrants on the left are identical. We need to calculate the following three products:

$$\begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 \\ \omega^0 & \omega^4 & \omega^0 & \omega^4 \\ \omega^0 & \omega^6 & \omega^4 & \omega^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_2 \\ a_4 \\ a_6 \end{bmatrix}, \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^1 & \omega^3 & \omega^5 & \omega^7 \\ \omega^2 & \omega^6 & \omega^2 & \omega^6 \\ \omega^3 & \omega^1 & \omega^7 & \omega^5 \end{bmatrix} \begin{bmatrix} a_1 \\ a_3 \\ a_5 \\ a_7 \end{bmatrix}, \begin{bmatrix} \omega^4 & \omega^4 & \omega^4 & \omega^4 \\ \omega^5 & \omega^7 & \omega^1 & \omega^3 \\ \omega^6 & \omega^2 & \omega^6 & \omega^2 \\ \omega^7 & \omega^5 & \omega^3 & \omega^1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_3 \\ a_5 \\ a_7 \end{bmatrix}$$

Calculating first product is exactly the same as calculating  $\text{DFT}(a_0, a_2, a_4, a_6)$ , so we can use recursion to calculate that. To calculate the others, observe that:

$$\begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^1 & \omega^3 & \omega^5 & \omega^7 \\ \omega^2 & \omega^6 & \omega^2 & \omega^6 \\ \omega^3 & \omega^1 & \omega^7 & \omega^5 \end{bmatrix} \begin{bmatrix} a_1 \\ a_3 \\ a_5 \\ a_7 \end{bmatrix} = \begin{bmatrix} \omega^0 & & & \\ & \omega^1 & & \\ & & \omega^2 & \\ & & & \omega^3 \end{bmatrix} \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 \\ \omega^0 & \omega^4 & \omega^0 & \omega^4 \\ \omega^0 & \omega^6 & \omega^4 & \omega^2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_3 \\ a_5 \\ a_7 \end{bmatrix}$$

$$\begin{bmatrix} \omega^4 & \omega^4 & \omega^4 & \omega^4 \\ \omega^5 & \omega^7 & \omega^1 & \omega^3 \\ \omega^6 & \omega^2 & \omega^6 & \omega^2 \\ \omega^7 & \omega^5 & \omega^3 & \omega^1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_3 \\ a_5 \\ a_7 \end{bmatrix} = \begin{bmatrix} \omega^4 & & & \\ & \omega^5 & & \\ & & \omega^6 & \\ & & & \omega^7 \end{bmatrix} \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 \\ \omega^0 & \omega^4 & \omega^0 & \omega^4 \\ \omega^0 & \omega^6 & \omega^4 & \omega^2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_3 \\ a_5 \\ a_7 \end{bmatrix}$$

That is, calculating the second and third product is the same as calculating  $\text{DFT}(a_1, a_3, a_5, a_7)$  and then multiplying the entries in the resulting vector. We now know that to calculate the transform of an array, it is enough to calculate the same transform for two arrays that are two times shorter, and then making a linear amount of extra calculations. Put together as an algorithm:

```
function fft (a[0], a[1], ..., a[m - 1]):
    if (m == 1):
        return (a[0])
    (e[0], e[1], ..., e[m / 2 - 1]) = fft(a[0], a[2], ..., a[m - 2])
    (o[0], o[1], ..., o[m / 2 - 1]) = fft(a[1], a[3], ..., a[m - 1])
    omega = exp(2 pi i / m)
    for (j = 0; j < m / 2; j++):
        c[j] = e[j] + omega^j o[j]
        c[j + m / 2] = e[j] - omega^j o[j] // same as e[j] + omega^(j + m / 2) o[j]
    return (c[0], c[1], ..., c[m - 1])
```

## Going the other way and putting it all together

Let's recap what we have so far. We want to calculate an array  $c$  with  $c_k = \sum_{i+j=k} a_i \cdot b_j$ . To do so, we need to:

- Pick a  $m = 2^k$  such that  $2n - 1 \leq m$ . Append zeros to  $a$  and  $b$  to make their length  $m$ .
- Using FFT, calculate  $\text{DFT}(a_0, a_1, \dots, a_{m-1}) = (A(\omega^0), \dots, A(\omega^{m-1}))$ .
- Using FFT, calculate  $\text{DFT}(b_0, b_1, \dots, b_{m-1}) = (B(\omega^0), \dots, B(\omega^{m-1}))$ .
- For each  $j$ , find  $C(\omega^j)$  by multiplying  $A(\omega^j)$  and  $B(\omega^j)$ .
- ???

To complete the algorithm, we need some way to take the tuple  $(C(\omega^0), C(\omega^1), \dots, C(\omega^{m-1}))$  and turn it back into the coefficients  $c_k$ . A sort of inverse operation for FFT. Luckily, FFT is kind-of-almost-self-dual, so it can be done by simply applying FFT again.

**Fact.**  $\text{DFT}(\text{DFT}(a_0, a_1, \dots, a_{m-1})) = (ma_0, ma_{m-1}, ma_{m-2}, \dots, ma_1)$ .

Proof

Define

$$(d_0, \dots, d_{m-1}) = \text{DFT}(c_0, \dots, c_{m-1})$$

$$(f_0, \dots, f_{m-1}) = \text{DFT}(d_0, \dots, d_{m-1}) = \text{DFT}(\text{DFT}(c_0, \dots, c_{m-1}))$$

Then

$$d_k = \sum_{j=0}^{m-1} c_j \omega^{kj}$$

and

$$f_\ell = \sum_{k=0}^{m-1} d_k \omega^{k\ell} = \sum_k \omega^{k\ell} \sum_j c_j \omega^{kj} = \sum_j c_j \sum_k \omega^{k(\ell+j)}.$$

If  $\ell + j \equiv 0 \pmod{m}$  then  $\omega^{k(\ell+j)} = \omega^0 = 1$  for each  $k$ . In all other cases,  
 $\sum_k \omega^{k(\ell+j)} = 0$ .

Proof

Denote  $\alpha = \omega^{\ell+j}$ . Observe that  $\alpha^m = \omega^{m(\ell+j)} = (\omega^m)^{\ell+j} = 1$ . We need to calculate  $\alpha^0 + \dots + \alpha^{m-1}$ . Since  $\ell + j \not\equiv 0 \pmod{m}$ , we have  $\alpha \neq 1$  and we can use the well-known formula:

$$\alpha^0 + \dots + \alpha^{m-1} = \frac{1-\alpha^m}{1-\alpha} = 0.$$

Therefore,  $f_\ell = mc_j$ , where  $j \equiv -\ell \pmod{m}$ .

Also, if we use the conjugate of  $\omega$  in place of  $\omega$  in the reverse transform, then we don't get this weird reordering. To summarize, the problem can be solved as follows:

- Pick a  $m = 2^k$  such that  $2n - 1 \leq m$ . Append zeros to  $a$  and  $b$  to make their length  $m$ .
- Using FFT, calculate  $\text{DFT}(a_0, a_1, \dots, a_{m-1}) = (A(\omega^0), \dots, A(\omega^{m-1}))$ .
- Using FFT, calculate  $\text{DFT}(b_0, b_1, \dots, b_{m-1}) = (B(\omega^0), \dots, B(\omega^{m-1}))$ .
- For each  $j$ , find  $C(\omega^j)$  by multiplying  $A(\omega^j)$  and  $B(\omega^j)$ .
- Using FFT, calculate  $\text{DFT}(C(\omega^0), \dots, C(\omega^{m-1})) = (mc_0, mc_{m-1}, \dots, mc_1)$ . Divide them all by  $m$  and reverse the tail.

## So, what is NTT?

And what is the deal with 998244353?

Think back to the algorithm. What properties of these powers of  $\omega$  did we actually use? Come to think of it, why did we use complex numbers at all? The only important property of  $\omega$  was that

$$\omega^m = 1$$

and that the numbers  $\omega^0, \omega^1, \dots, \omega^{m-1}$  were all distinct. The technical term for such a number is a  $m$ -th *primitive root of unity*. For real numbers, such  $\omega$  don't exist if  $m > 2$  (that is, pretty much always). For complex numbers, they always exist as seen above.

What about other kinds of number systems? In competitive programming, especially in counting problems, we often don't work with neither real nor complex numbers. Instead, we calculate the number of things modulo some large prime. Remember that modulo  $p$ , some polynomial equations are solvable even though they aren't solvable in the real numbers. For example, there is no real number  $x$  satisfying  $x^2 = -1$ , but modulo 13 there are:  $5^2 = 25 \equiv -1 \pmod{13}$ . What about solving  $\omega^m = 1$ ?

**Fact.** A  $m$ -th primitive root of unity exists modulo a prime  $p$  if and only if  $p - 1$  is divisible by  $m$ .

Now look at this.

$$998244353 = 2^{23} \cdot 7 \cdot 11 + 1.$$

A primitive  $m$ -th root of unity will exist for any small enough power of two. This means that modulo 998244353, we can calculate FFTs of length up to  $2^{23}$ , which is enough for almost all competitive programming applications. This variation of FFT is sometimes called the *number theoretic transform* or NTT.

## Applications

If you look at hard problems, especially on longer contests, you can find a lot of complicated applications of FFT. For the more involved ones, it is useful to think of [generating functions](#) and [whatever this is](#). Sometimes this means applying other [polynomial operations](#), almost all of which use FFT as a subroutine.

Brief rant

In blogs like this, I dislike talking about heavy abstractions or advanced topics that go far beyond the thing the blog was actually supposed to teach; e.g. I don't think it is appropriate to introduce things like the Chirp Z-transform or advanced theorems about generating functions.

People in general learn by moving from concrete to abstract. That is, for most people it is best to first learn "normal" FFT. Once they are comfortable with it, generalizations and abstractions make more sense.

However, in this blog let's stick to the basics and first try to understand why FFT comes up. I mentioned right at the beginning that this type of expression is common in formulas for counting something.

**Problem.** (made up educational problem) There is a game that up to  $n$  players can play. Players play in two teams: Team Red and Team Blue. You have figured out that if there are  $i$  players in Team Red, Team Red has a probability  $p[i]$  of winning. The array  $p$  has length  $n + 1$  and is given in the input. Players are assigned to teams randomly. For each  $k = 1, \dots, n$ , calculate the probability that Team Red wins if there are  $k$  players.  $n \leq 2 \cdot 10^5$ .

Solution

For a given  $k$ , the answer is clearly

$$\frac{1}{2^k} \sum_{i=0}^k \binom{k}{i} p[i] = \frac{k!}{2^k} \sum_{i=0}^k \frac{p[i]}{i!} \cdot \frac{1}{(k-i)!}.$$

Substitute  $j = k - i$  and the formula becomes

$$\frac{k!}{2^k} \sum_{i+j=k} \frac{p[i]}{i!} \cdot \frac{1}{j!}.$$

We need to calculate this for every  $k$ . If we define the arrays  $a$  and  $b$  by  $a[i] = \frac{p[i]}{i!}$  and  $b[j] = \frac{1}{j!}$ , then the right part of the formula becomes exactly  $\sum_{i+j=k} a[i] \cdot b[j]$ . Calculate these with FFT and then multiply the  $k$ -th value by  $\frac{k!}{2^k}$ .

## Cross-correlation

It's very common to see this simple variation: instead of  $c[k] = \sum_{i+j=k} a[i] \cdot b[j]$ , calculate  $c[k] = \sum_{i-j=k} a[i] \cdot b[j]$ . A sum of this type is sometimes called *cross-correlation*.

**Problem.** (CSA Round #75 F) Given  $n$  and  $q$  queries. Each query consists of two integers  $1 \leq x < y \leq n$ . For each query, calculate modulo 998244353 the number of permutations  $p$  of length  $n$  such that  $p[y] = \max_{i=1}^y p[i]$  and  $2 \cdot p[x] < p[x]$ .

Solution

For fixed  $x$  and  $y$ , the answer is

$$(n-y)! \sum_{u \geq y} \frac{(u-2)!}{(u-y)!} \left\lfloor \frac{u-1}{2} \right\rfloor.$$

Here, we treat the factorial of a negative number as zero.

Derivation, but try to think about it yourself for a bit

Think of iterating over all possible values of  $p[y]$ . Let  $u = p[y]$ . There are:

- $\left\lfloor \frac{u-1}{2} \right\rfloor$  ways to choose  $p[x]$ .
- $\frac{(u-2)!}{(u-y)!} = \binom{u-2}{y-2} (y-2)!$  ways to choose values for positions 1 through  $y-1$  except for  $x$ . There are  $y-2$  positions to fill and we have  $u-2$  valid values for them: 1 through  $u-1$  except for  $p[x]$ .
- $(n-y)!$  ways to choose an order for the rest of the elements.

Sum them all up. The  $(n-y)!$  term does not depend on  $u$  and can be brought in front of the sum sign.

Notice that this expression doesn't depend on  $x$  at all (try to convince yourself that this is not weird). It now makes sense to calculate the answer for all  $y$  immediately after reading  $n$ .

Substitute  $i = u$ ,  $j = u - y$ ,  $k = y$ . The answer becomes:

$$(n-k)! \sum_{i-j=k} \frac{(i-2)!}{j!} \left\lfloor \frac{i-1}{2} \right\rfloor.$$

Define arrays  $a$  and  $b$  of length  $n + 1$  by

$$a[i] = (i-2)! \left\lfloor \frac{i-1}{2} \right\rfloor, \quad b[j] = \frac{1}{j!}.$$

Now the expression we have to calculate becomes exactly  $\sum_{i-j=k} a[i] \cdot b[j]$ . To calculate it, simply reverse  $b$  and calculate the convolution with FFT as above. The result will be what we want, shifted by the length of the reversed array.



## Fuzzy search

A common application of FFT is comparing a string to all substrings of another string. One specific example where you can apply that is [528D - Fuzzy Search](#), but the idea is older than that.

**Problem.** (Classical) Given a longer string  $s$  (of length  $n$ ) and a shorter string  $t$  (of length  $m$ ) over a relatively small alphabet. For each  $i$ , calculate the similarity of  $s[i \dots i + m - 1]$  and  $t$ . The similarity of two strings is defined as the number of positions where they have the same character. For example, the similarity of "abbaa" and "aabba" is 3, because they match on positions 0, 2 and 4.  $m < n \leq 10^5$ .

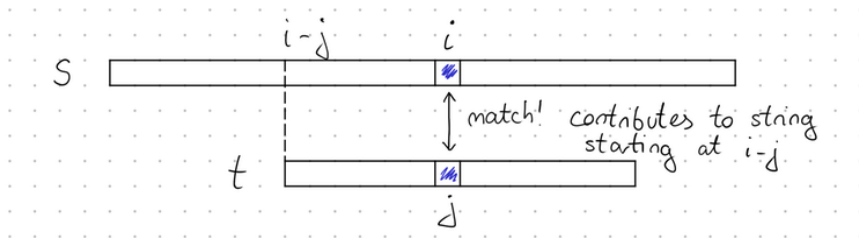
Solution

Suppose that the strings just consist of "a" and "b". We show how to compute, for each  $i$ , the number of matching "a"-s in  $s[i \dots i + m - 1]$  and  $t$ .

We define arrays  $s_a$  and  $t_a$  of length  $n$  and  $m$ , respectively.

$$s_a[i] = \begin{cases} 1 & \text{if } s[i] = \text{a} \\ 0 & \text{otherwise} \end{cases} \quad t_a[i] = \begin{cases} 1 & \text{if } t[i] = \text{a} \\ 0 & \text{otherwise} \end{cases}$$

If  $s[i]$  and  $t[j]$  are both "a", then  $s_a[i] \cdot t_a[j] = 1$ . In this case, they constitute a match between  $s[i - j \dots i - j + m - 1]$  and  $t$ . In all other cases,  $s_a[i] \cdot t_a[j] = 0$ .



That is, the number of matching "a"-s in the string  $s[k \dots k + m - 1]$  is

$$\sum_{i-j=k} s_a[i] \cdot t_a[j].$$

We must calculate this for all  $k$  and have already learned how to efficiently calculate sums of this type.

## Subset sum variations

Suppose you have  $n$  rocks and the  $i$ -th of them weighs  $a_i$  grams. How many ways are there to pick rocks such that their total weight is exactly  $k$ ? Depending on the constraints and the exact problem, there are many ways to think about this. But one way is to model them is with polynomials. The answer to the question is exactly the coefficient of  $x^k$  in the polynomial

$$(1 + x^{a_0})(1 + x^{a_1})(1 + x^{a_2}) \cdots (1 + x^{a_{n-1}}).$$

Here is one problem where a similar perspective is useful.

**Problem.** (1096G - Lucky Tickets) We consider decimal numbers with even length  $n$ , potentially with leading zeroes. You are given a subset of decimal digits that are allowed. Count, modulo 998244353, the number of numbers with length  $n$  that consist of only allowed digits and whose first  $n/2$  digits sum to the same value as the last  $n/2$  digits.  $n \leq 2 \cdot 10^5$ .

Solution

Let  $p(x) = p_0x^0 + p_1x^1 + \cdots + p_9x^9$  where  $p_d = 1$  if  $d$  is allowed and 0 otherwise. The number of ways to make  $n/2$  digits sum to  $k$  is the coefficient of  $x^k$  in  $(p(x))^{n/2}$ . The polynomial  $(p(x))^{n/2}$  can be calculated by using FFT and fast exponentiation.

Thank you for reading! Please let me know how you liked it. I'm especially curious to hear what people think about hand-drawn figures.

▲ +436 ▼ ☆

👤 -is-this-fft-

📅 3 years ago

💬 32

```


1  #include<bits/stdc++.h>
2  using namespace std;
3
4  using Complex = complex<double>;
5  void fft(vector<Complex>& a) {
6      static vector<complex<long double>> R(2, 1);
7      static vector<Complex> rt(2, 1); // (^ 10% faster if double)
8      int n = (int)a.size(), L = 31 - __builtin_clz(n);
9      for(static int k = 2; k < n; k *= 2) {
10         R.resize(n);
11         rt.resize(n);
12         auto x = polar(1.0L, acos(-1.0L) / k);
13         for (int i = k; i < 2 * k; ++i) {
14             rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
15         }
16     }
17     vector<int> rev(n);
18     for(int i = 0; i < n; ++i) {
19         rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
20         if (i < rev[i]) swap(a[i], a[rev[i]]);
21     }
22     for (int k = 1; k < n; k *= 2)
23         for (int i = 0; i < n; i += 2 * k)
24             for (int j = 0; j < k; ++j) {
25                 // Complex z = rt[j+k] * a[i+j+k]; // (25% faster if hand-rolled) /// inclu
26                 auto x = (double *)&rt[j+k], y = (double *)&a[i+j+k]; /// excl
27                 Complex z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]); /// excl
28                 a[i + j + k] = a[i + j] - z;
29                 a[i + j] += z;
30             }
31 }
32
33 template<typename T = long long>
34 vector<T> convolution(const vector<T>& a, const vector<T>& b) {
35     if (a.empty() || b.empty()) return {};
36     vector<T> res((int)a.size() + (int)b.size() - 1);
37     int L = 32 - __builtin_clz((int)res.size()), n = 1 << L;
38     vector<Complex> in(n), out(n);
39     copy(a.begin(), a.end(), in.begin());
40     for (int i = 0; i < (int)b.size(); ++i) in[i].imag(b[i]);
41     fft(in);
42     for (Complex& x: in) x *= x;
43     for (int i = 0; i < n; ++i) out[i] = in[-i & (n - 1)] - conj(in[i]);
44     fft(out);
45     for (int i = 0; i < (int)res.size(); ++i) {
46         res[i] = (T)llround(imag(out[i]) / (4 * n)); // remove rounding if the output i
47     }
48     return res;
49 }
50 const int MAX_DEG = 2e6 + 10;
51
52 int main() {
53     ios_base::sync_with_stdio(false);
54     cin.tie(nullptr);
55     int n;
56     cin >> n;
57     vector<long long> s(MAX_DEG);
58
59     for(int i = 0; i < n; ++i) {
60         int x;
61         cin >> x;
62         s[x] += 1;

```

2025-11-04 (Tue)  
17:34:17 -03:00

```
63 }
64 while(!s.empty() && s.back() == 0) s.pop_back();
65 auto ss = convolution(s, s);
66 long long ans = 0;
67 // cerr << ss.size() << ' ' << s.size() << '\n';
68 for(int i = 0, len = (int)ss.size(); i < len; i += 2) {
69     long long c = (ss[i] - s[i / 2]) / 2;
70     ans += c * s[i / 2];
71     // cout << "sum " << i << ": " << ss[i] << ' ' << s[i] << ' ' << c << '\n';
72 }
73 cout << ans << '\n';
74
75 return 0;
76 }
```







Submission Info

Submission Time	2025-10-01 22:17:42
Task	G - Fine Triplets (/contests/abc392/tasks/abc392_g)
User	gabmei (/users/gabmei)  (/contests/abc392/submissions?f.User=gabmei)
Language	C++ 20 (gcc 12.2)
Score	600
Code Size	2467 Byte
Status	<div>AC</div>
Exec Time	510 ms
Memory	206840 KiB





Contest Duration: 2025-02-08(Sat) 09:00 (<http://www.timeanddate.com/worldclock/fixedtime.html?iso=20250208T2100&p1=248>) - 2025-02-08(Sat) 10:40 (<http://www.timeanddate.com/worldclock/fixedtime.html?iso=20250208T2100&p1=248>) (local time) (100 minutes)

[Back to Home \(/home\)](#)

Judge Result


 <a href="#">Top (/contests/abc392)</a>	 <a href="#">Tasks (/contests/abc392/tasks)</a>	All
 <a href="#">Clarifications (/contests/abc392/clarifications)</a>	 <a href="#">Submit (/contests/abc392/submit)</a>	600 / 600
 <a href="#">Results</a>	 <a href="#">Standings (/contests/abc392/standings)</a>	<div>AC</div> × 43

 [Virtual Standings \(/contests/abc392/standings/virtual\)](#)

Set Name	Test Cases
 <a href="#">Custom Test (/contests/abc392/custom_test)</a>	 <a href="#">Editorial (/contests/abc392/editorial)</a>
<a href="#">Sample</a>	sample_01.txt, sample_02.txt, sample_03.txt
 <a href="https://codeforces.com/blog/entry/139254">Discuss (https://codeforces.com/blog/entry/139254)</a>	
All	sample_01.txt, sample_02.txt, sample_03.txt, test_01.txt, test_02.txt, test_03.txt, test_04.txt, test_05.txt, test_06.txt, test_07.txt, test_08.txt, test_09.txt, test_10.txt, test_11.txt, test_12.txt, test_13.txt, test_14.txt, test_15.txt, test_16.txt, test_17.txt, test_18.txt, test_19.txt, test_20.txt, test_21.txt, test_22.txt, test_23.txt, test_24.txt, test_25.txt, test_26.txt, test_27.txt, test_28.txt, test_29.txt, test_30.txt, test_31.txt, test_32.txt, test_33.txt, test_34.txt, test_35.txt, test_36.txt, test_37.txt, test_38.txt, test_39.txt, test_40.txt
	2025-11-04 (Tue) 17:34:17 -03:00

[HOME](#) [TOP](#) [CATALOG](#) [CONTESTS](#) [GYM](#) [PROBLEMSET](#) [GROUPS](#) [RATING](#) [EDU](#) [API](#) [CALENDAR](#) [HELP](#) [RAYAN](#) 
[PROBLEMS](#) [SUBMIT CODE](#) [MY SUBMISSIONS](#) [STATUS](#) [STANDINGS](#) [CUSTOM INVOCATION](#)

General

#	Author	Problem	Lang	Verdict	Time	Memory	Sent	Judged		
343781685	Practice: <a href="#">pastilia</a>	<a href="#">584176I</a> - 35	C++23 (GCC 14-64, msys2)	Accepted	358 ms	249976 KB	2025-10- 15 03:35:40	2025-10- 15 03:35:40		<a href="#">Compare</a>

[→ Source](#)[Copy](#)

```
#include <bits/stdc++.h>

// #define MOD 1000000007
#define INF 0x3f3f3f3f
#define INFLl 0x3f3f3f3f3f3f3fll
#define sz(x) (int)x.size()
#define s second
#define f first

typedef long long ll;
typedef unsigned long long ull;

#define MAX 200005
#define MAXLOG 35

using namespace std;

// Aho-corasick
//
// query retorna o somatorio do numero de matches de
// todas as stringuinhas na stringona
//
// insert - O(|s| log(SIGMA))
// build - O(N), onde N = somatorio dos tamanhos das strings
// query - O(|s|)

vector<int> ocs[MAX];

namespace aho {
map<char, int> to[MAX];
int link[MAX], idx, term[MAX], savet[MAX], exit[MAX], sobe[MAX];

void insert(string& s, int i) {
    int at = 0;
    for (char c : s) {
        auto it = to[at].find(c);
        if (it == to[at].end())
            at = to[at][c] = ++idx;
        else
            at = it->second;
    }
    term[at]++, savet[at] = i, sobe[at]++;
}

void build() {
    queue<int> q;
    q.push(0);
    link[0] = exit[0] = -1;
    while (q.size()) {
        int i = q.front();
        q.pop();
        for (auto [c, j] : to[i]) {
            int l = link[i];
            while (l != -1 and !to[l].count(c)) l = link[l];
            link[j] = l == -1 ? 0 : to[l][c];
            exit[j] = term[link[j]] ? link[j] : exit[link[j]];
            if (exit[j] + 1) sobe[j] += sobe[exit[j]];
            q.push(j);
        }
    }
}

void query(string& s) {
    int at = 0;
    for (int i = 0; i < sz(s); i++) {
        char c = s[i];
        while (at != -1 and !to[at].count(c)) at = link[at];
        at = at == -1 ? 0 : to[at][c];
        // find occurences
        int curat = at;
        if (!term[curat]) curat = exit[curat];
        while (curat != -1) {
            ocs[savet[curat]].push_back(i);
            curat = exit[curat];
        }
    }
}
```

```

}
} // namespace aho

void solve() {
    string s;
    cin >> s;
    int n;
    cin >> n;
    vector<int> k(n);
    vector<string> t(n);
    for (int i = 0; i < n; i++) {
        cin >> k[i] >> t[i];
        aho::insert(t[i], i);
    }
    aho::build();
    aho::query(s);
    for (int i = 0; i < n; i++) {
        if (sz(ocs[i]) < k[i]) {
            cout << "-1\n";
            continue;
        }
        int mini = INF;
        for (int j = k[i] - 1; j < sz(ocs[i]); j++) {
            mini = min(mini, ocs[i][j] - ocs[i][j - (k[i] - 1)] + 1 + sz(t[i]) - 1);
        }
        cout << mini << endl;
    }
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    int tt = 1;
    // cin >> tt;
    while (tt--) {
        solve();
    }
    return 0;
}

```

## Judged Example Tests

1

**Time:** 46 ms, **memory:** 18016 KB

**Verdict:** OK

**Input**

```

aaaaa
5
3 a
3 aa
2 aaa
3 aaaa
1 aaaaa

```

**Participant's output**

```

3
4
4
-1
5

```

**Checker comment**

ok 5 number(s): "3 4 4 -1 5"

2

**Time:** 46 ms, **memory:** 18016 KB

**Verdict:** OK

**Input**

```

abbbb
7
4 b
1 ab
3 bb
1 abb
2 bbb
1 a
2 abbb

```

**Participant's output**

```


-1
2
-1
3
-1
1
-1

```

**Checker comment**

ok 7 numbers

General

#	Author	Problem	Lang	Verdict	Time	Memory	Sent	Judged		
303589016	Practice: <a href="#">pastilia</a>	<a href="#">584176I</a> - 35	C++17 (GCC 7-32)	Accepted	609 ms	60 KB	2025-01-30 03:49:03	2025-01-30 03:49:03		<a href="#">Compare</a>

[→ Source](#)[Copy](#)

```
#include <bits/stdc++.h>

#pragma GCC optimize("Ofast")
#pragma GCC target("fma,sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,avx2,tune=native")
#pragma GCC optimize("unroll-loops")

#define MAX 200005
// #define MOD 998244353
#define INF 0x3f3f3f3f
#define INFL 0x3f3f3f3f3f3f3fll
#define sz(x) (int)x.size()
#define se second
#define fi first

typedef long long ll;
typedef unsigned long long ull;

using namespace std;

// Suffix Array - O(n log n)
//
// kasai recebe o suffix array e calcula lcp[i],
// o lcp entre s[sa[i],...,n-1] e s[sa[i+1],...,n-1]
//
// Complexidades:
// suffix_array - O(n log(n))
// kasai - O(n)

vector<int> suffix_array(string s) {
    s += "$";
    int n = s.size(), N = max(n, 260);
    vector<int> sa(n), ra(n);
    for(int i = 0; i < n; i++) sa[i] = i, ra[i] = s[i];

    for(int k = 0; k < n; k ? k *= 2 : k++) {
        vector<int> nsa(sa), nra(n), cnt(N);

        for(int i = 0; i < n; i++) nsa[i] = (nsa[i]-k+n)%n, cnt[ra[i]]++;
        for(int i = 1; i < N; i++) cnt[i] += cnt[i-1];
        for(int i = n-1; i+1; i--) sa[--cnt[ra[nsa[i]]]] = nsa[i];

        for(int i = 1, r = 0; i < n; i++) nra[sa[i]] = r += ra[sa[i]] !=
            ra[sa[i-1]] or ra[(sa[i]+k)%n] != ra[(sa[i-1]+k)%n];
        ra = nra;
        if (ra[sa[n-1]] == n-1) break;
    }
    return vector<int>(sa.begin()+1, sa.end());
}

vector<int> kasai(string s, vector<int> sa) {
    int n = s.size(), k = 0;
    vector<int> ra(n), lcp(n);
    for (int i = 0; i < n; i++) ra[sa[i]] = i;

    for (int i = 0; i < n; i++, k = !k) {
        if (ra[i] == n-1) { k = 0; continue; }
        int j = sa[ra[i]+1];
        while (i+k < n and j+k < n and s[i+k] == s[j+k]) k++;
        lcp[ra[i]] = k;
    }
    return lcp;
}

void solve(){
    string s;
    int m;
    cin>>s>>m;
    int n=sz(s);
    vector<int>sa=suffix_array(s);
    vector<int>lcp=kasai(s,sa);
    // for(int i=0;i<n;i++){
    //     cout<<i<<": ";
    //     cout<<sa[i]<<" ";
    //     cout<<s.substr(sa[i],sz(s)-sa[i])<<endl;
    // }
```

```

for(int i=0;i<m;i++){
    int k;
    string t;
    cin>>k>>t;
    // cout<<k<<" "<<t<<" ";
    // find first occ of t
    int l=0,r=n;
    int first_occ=-1;
    while(l<=r){
        int mb=l+(r-l)/2;
        // cout<<l<<" "<<r<<" "<<mb<<endl;
        int deu=(n-sa[mb])>=sz(t); // 0: <, 1: =, 2: >
        for(int c=sa[mb];c<n and c-sa[mb]<sz(t);c++){ // sum of all queries is LE than 10e5
            if(s[c]<t[c-sa[mb]]){
                deu=0;break;
            }
            if(s[c]>t[c-sa[mb]]){
                deu=2;break;
            }
        }
        if(deu>=1){
            if(deu==1)first_occ=mb;
            r=mb-1;
        }
        else{
            l=mb+1;
        }
    }
    // find last occ of t
    l=0,r=n-1;
    int last_occ=-1;
    while(l<=r){
        int mb=l+(r-l)/2;
        int deu=1; // 0: <, 1: =, 2: >
        for(int c=sa[mb];c<n and c-sa[mb]<sz(t);c++){ // sum of all queries is LE than 10e5
            if(s[c]<t[c-sa[mb]]){
                deu=0;break;
            }
            if(s[c]>t[c-sa[mb]]){
                deu=2;break;
            }
        }
        if(deu<=1){
            if(deu==1)last_occ=mb;
            l=mb+1;
        }
        else{
            r=mb-1;
        }
    }
    // cout<<first_occ<<" "<<last_occ<<endl;
    // sweep line to find smallest difference if there is k
    vector<int>swp;
    for(int j=first_occ;j>=0 and j<=last_occ;j++){
        swp.push_back(sa[j]);
    }
    sort(swp.begin(),swp.end());
    // for(int j=0;j<sz(swp);j++){
    //     cout<<swp[j]<<" ";
    // }
    // cout<<endl;
    if(sz(swp)<k){
        cout<<"-1\n";
        continue;
    }
    int mini=INF;
    for(int j=k-1;j<sz(swp);j++){
        mini=min(mini,swp[j]-swp[j-(k-1)]+1+sz(t)-1);
    }
    cout<<mini<<endl;
}
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int tt=1;
    // cin>>tt;
    while(tt--){
        solve();
    }
    return 0;
}

```

## Judged Example Tests

1

Time: 15 ms, memory: 0 KB

Verdict: OK

Input


```

aaaaa
5
3 a
3 aa
2 aaa

```

[HOME](#) [TOP](#) [CATALOG](#) [CONTESTS](#) [GYM](#) [PROBLEMSET](#) [GROUPS](#) [RATING](#) [EDU](#) [API](#) [CALENDAR](#) [HELP](#) [RAYAN](#) 
[PROBLEMS](#) [SUBMIT CODE](#) [MY SUBMISSIONS](#) [STATUS](#) [STANDINGS](#) [CUSTOM INVOCATION](#)

General

#	Author	Problem	Lang	Verdict	Time	Memory	Sent	Judged		
303555979	Practice: <a href="#">pastilia</a>	<a href="#">584381I</a> - 19	C++17 (GCC 7-32)	Accepted	93 ms	244676 KB	2025-01- 29 20:51:44	2025-01- 29 20:51:44		<a href="#">Compare</a>

[→ Source](#)[Copy](#)

```
#include <bits/stdc++.h>

#pragma GCC optimize("Ofast")
#pragma GCC target("fma,sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,avx2,tune=native")
#pragma GCC optimize("unroll-loops")

#define MAX 200005
#define MOD 1000000007
#define INF 0x3f3f3f3f
#define INFLl 0x3f3f3f3f3f3f3fll
#define sz(x) (long long)x.size()
#define se second
#define fi first

#define N_CELL 4
#define M_CELL 3

typedef long long ll;
typedef unsigned long long ull;

using namespace std;

int nc,ns,nt;
string c,s,t;
vector<int> pis(1000),pit(50);

void construct_pis() {
    for (int i = 1, j = 0; i < s.size(); i++) {
        while (j and s[j] != s[i]) j = pis[j-1];
        if (s[j] == s[i]) j++;
        pis[i] = j;
    }
}

void construct_pit() {
    for (int i = 1, j = 0; i < t.size(); i++) {
        while (j and t[j] != t[i]) j = pit[j-1];
        if (t[j] == t[i]) j++;
        pit[i] = j;
    }
}

int dp[1000][1000][50];
bool seen[1000][1000][50];

int p(int idx,int spi,int tpi){
    // cout<<idx<<" "<<spi<<" "<<tpi<<endl;
    if(idx==nc){
        return 0;
    }
    // *a*
    // bba
    // b

    if(seen[idx][spi][tpi]){
        return dp[idx][spi][tpi];
    }
    seen[idx][spi][tpi]=true;
    if(c[idx]!='*'){
        // kmp s
        int curs=spi;
        while(curs and s[curs]!=c[idx])curs=pis[curs-1];
        if(s[curs]==c[idx])curs++;
        // kmp t
        int curt=tpi;
        while(curt and t[curt]!=c[idx])curt=pit[curt-1];
        if(t[curt]==c[idx])curt++;
        return dp[idx][spi][tpi]=p(idx+1,curs,curt)+(curs==ns)-(curt==nt);
    }
}

int ans=-INF;
for(int i=0;i<26;i++){
    // kmp s
    int curs=spi;
    while(curs and s[curs]!=(i+'a'))curs=pis[curs-1];
    if(s[curs]==(i+'a'))curs++;
    // kmp t
```

```

        int curt=tpi;
        while(curt and t[curt]!=(i+'a'))curt=pit[curt-1];
        if(t[curt]==(i+'a'))curt++;
        ans=max(ans,p(idx+1,curs,curt)+(curs==ns)-(curt==nt));
    }
    return dp[idx][spi][tpi]=ans;
}

void solve(){
    cin>>c>>s>>t;
    memset(seen,0,sizeof(seen));
    if(sz(s)>sz(c)){
        s="";
        for(int i=0;i<sz(c);i++)s+='#';
    }
    nc=sz(c),ns=sz(s),nt=sz(t);
    construct_pis();
    construct_pit();
    cout<<p(0,0,0)<<endl;
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int tt=1;
    // cin>>tt;
    while(tt--){
        solve();
    }
    return 0;
}

```

## Judged Example Tests

1

**Time:** 61 ms, **memory:** 244620 KB

**Verdict:** OK

**Input**

\*\*\*\*\*  
katie  
shiro

**Participant's output**

1

**Checker comment**

ok 1 number(s): "1"

2

**Time:** 46 ms, **memory:** 244660 KB

**Verdict:** OK

**Input**

caat  
caat  
a

**Participant's output**

-1

**Checker comment**

ok 1 number(s): "-1"

3

**Time:** 62 ms, **memory:** 244628 KB

**Verdict:** OK

**Input**

\*a\*  
bba  
b

**Participant's output**

0

**Checker comment**

ok 1 number(s): "0"

4

**Time:** 62 ms, **memory:** 244624 KB

**Verdict:** OK

**Input**

\*\*\*  
cc  
z

**Participant's output**

2

**Checker comment**

ok 1 number(s): "2"