多项式

cxy2022

OIers' Chat

12nd August, 2022

Contents

约定

约定

- 本文中的代码的语言都是 c++。

约定

- 本文中的代码的语言都是 c++。
- 本文中的多项式都是一元多项式，而且不会去区分多项式和单项式，所以实际上本文中提到的多项式指的是一元整式。

约定

- 本文中的代码的语言都是 c++。
- 本文中的多项式都是一元多项式，而且不会去区分多项式和单项式，所以实际上本文中提到的多项式指的是一元整式。
- 如果不加特殊说明，本文中的 n 和 m 指的都是多项式的次数。

约定

- 本文中的代码的语言都是 c++。
- 本文中的多项式都是一元多项式，而且不会去区分多项式和单项式，所以实际上本文中提到的多项式指的是一元整式。
- 如果不加特殊说明，本文中的 n 和 m 指的都是多项式的次数。
- 如果不加特殊说明，本文中的复杂度指的都是时间复杂度。

约定

- 本文中的代码的语言都是 c++。
- 本文中的多项式都是一元多项式，而且不会去区分多项式和单项式，所以实际上本文中提到的多项式指的是一元整式。
- 如果不加特殊说明，本文中的 n 和 m 指的都是多项式的次数。
- 如果不加特殊说明，本文中的复杂度指的都是时间复杂度。
- 如果不加特殊说明，本文中的 p 都是质数。

Contents

多项式加减法

多项式加减法

这个没什么可以说的吧，对应项加减即可，复杂度线性。

Contents

多项式乘法

多项式乘法

朴素地去进行竖式乘法的复杂度是 $O(nm)$，并不能接受。

$O\left(n^{\log_2 3}\right)$ 的分治算法

$O\left(n^{\log_2 3}\right)$ 的分治算法

假设 $n \geq m$，我们先把 m 次的那个多项式在高次补 0 补到 n 次，

$O\left(n^{\log_2 3}\right)$ 的分治算法

假设 $n \geq m$，我们先把 m 次的那个多项式在高次补 0 补到 n 次，然后接下来要求的就是两个次数都是 n 的多项式的乘积。

$O\left(n^{\log_2 3}\right)$ 的分治算法

假设 $n \geq m$，我们先把 m 次的那个多项式在高次补 0 补到 n 次，然后接下来要求的就是两个次数都是 n 的多项式的乘积。
而且注意到我们可以只用 3 次实数乘法以及若干次实数加法计算出两个一次多项式 $ax+b$、$cx+d$ 的乘积！

$O\left(n^{\log_2 3}\right)$ 的分治算法

假设 $n \geq m$，我们先把 m 次的那个多项式在高次补 0 补到 n 次，然后接下来要求的就是两个次数都是 n 的多项式的乘积。
而且注意到我们可以只用 3 次实数乘法以及若干次实数加法计算出两个一次多项式 $ax+b$、$cx+d$ 的乘积！
令 $e=ac$，$f=bd$，$g=(a+b)(c+d)$，可以发现 $(ax+b)(cx+d)=ex^2+(g-e-f)x+f$，所以我们只需要使用 3 次实数乘法计算出 e、f 和 g 即可！

$O\left(n^{\log_2 3}\right)$ 的分治算法

$O\left(n^{\log_2 3}\right)$ 的分治算法

然后回到上面说的求两个 n 次多项式的积

$O\left(n^{\log_2 3}\right)$ 的分治算法

然后回到上面说的求两个 n 次多项式的积
假设它们分别是 $f(x)$ 和 $g(x)$，并求出 4 个多项式 $A(x)$、(x)、$C(x)$ 和 $D(x)$，

$O\left(n^{\log_2 3}\right)$ 的分治算法

然后回到上面说的求两个 n 次多项式的积
假设它们分别是 $f(x)$ 和 $g(x)$，并求出 4 个多项式 $A(x)$、(x)、$C(x)$ 和 $D(x)$，
使得

$O\left(n^{\log_2 3}\right)$ 的分治算法

然后回到上面说的求两个 n 次多项式的积
假设它们分别是 $f(x)$ 和 $g\left(x\right)$，并求出 4 个多项式 $A\left(x\right)$、(x)、$C\left(x\right)$ 和 $D\left(x\right)$，
使得
$f(x)=A\left(x\right)x^{\left\lfloor\frac{n+1}{2}\right\rfloor}+B\left(x\right)$,

$O\left(n^{\log_2 3}\right)$ 的分治算法

然后回到上面说的求两个 n 次多项式的积
假设它们分别是 $f(x)$ 和 $g(x)$，并求出 4 个多项式 $A(x)$、(x)、$C(x)$ 和 $D(x)$，
使得
$f(x) = A(x)\,x^{\lfloor\frac{n+1}{2}\rfloor} + B(x)$,
$g(x) = C(x)\,x^{\lfloor\frac{n+1}{2}\rfloor} + D(x)$,

$O\left(n^{\log_2 3}\right)$ 的分治算法

然后回到上面说的求两个 n 次多项式的积
假设它们分别是 $f(x)$ 和 $g(x)$，并求出 4 个多项式 $A(x)$、(x)、$C(x)$ 和 $D(x)$，
使得
$f(x) = A(x)\, x^{\left\lfloor \frac{n+1}{2} \right\rfloor} + B(x)$，
$g(x) = C(x)\, x^{\left\lfloor \frac{n+1}{2} \right\rfloor} + D(x)$，
然后套用上面的用 3 次实数计算一次多项式的乘积的做法，

$O\left(n^{\log_2 3}\right)$ 的分治算法

然后回到上面说的求两个 n 次多项式的积
假设它们分别是 $f(x)$ 和 $g(x)$，并求出 4 个多项式 $A(x)$、(x)、$C(x)$ 和 $D(x)$，
使得
$f(x)=A(x)\,x^{\lfloor\frac{n+1}{2}\rfloor}+B(x)$,
$g(x)=C(x)\,x^{\lfloor\frac{n+1}{2}\rfloor}+D(x)$,
然后套用上面的用 3 次实数计算一次多项式的乘积的做法，
就只需要进行 3 次不超过 $\lceil\frac{n+1}{2}\rceil$ 次多项式的乘法

$O\left(n^{\log_2 3}\right)$ 的分治算法

然后回到上面说的求两个 n 次多项式的积
假设它们分别是 $f(x)$ 和 $g(x)$，并求出 4 个多项式 $A(x)$、(x)、$C(x)$ 和 $D(x)$，
使得
$f(x) = A(x)\,x^{\lfloor\frac{n+1}{2}\rfloor} + B(x)$,
$g(x) = C(x)\,x^{\lfloor\frac{n+1}{2}\rfloor} + D(x)$,
然后套用上面的用 3 次实数计算一次多项式的乘积的做法，
就只需要进行 3 次不超过 $\lceil\frac{n+1}{2}\rceil$ 次多项式的乘法
以及常数次不超过 $\lceil\frac{n+1}{2}\rceil$ 次多项式的加法就可以计算出两个 n 次多项式的乘积！

$O\left(n^{\log_2 3}\right)$ 的分治算法

然后回到上面说的求两个 n 次多项式的积
假设它们分别是 $f(x)$ 和 $g(x)$，并求出 4 个多项式 $A(x)$、(x)、$C(x)$ 和 $D(x)$，
使得
$f(x) = A(x)\,x^{\lfloor \frac{n+1}{2} \rfloor} + B(x)$，
$g(x) = C(x)\,x^{\lfloor \frac{n+1}{2} \rfloor} + D(x)$，
然后套用上面的用 3 次实数计算一次多项式的乘积的做法，
就只需要进行 3 次不超过 $\lceil \frac{n+1}{2} \rceil$ 次多项式的乘法
以及常数次不超过 $\lceil \frac{n+1}{2} \rceil$ 次多项式的加法就可以计算出两个 n 次多项式的乘积！
$T(n) = 3T\left(\frac{n}{2}\right) + O(n)$，根据主定理可以得到
$T(n) = O\left(n^{\log_2 3}\right)$。

$O\left(n\log n\right)$ 的快速傅里叶变换算法

$O\left(n\log n\right)$ 的快速傅里叶变换算法

首先有个东西叫做 n 次单位根 ω_n，是一个满足 $\omega_n^n = 1$ 的复数

$O(n\log n)$ 的快速傅里叶变换算法

首先有个东西叫做 n 次单位根 ω_n，是一个满足 $\omega_n^n=1$ 的复数
只不过我们在这里要求，不能存在一个比 n 小的 m 使得 ω_n 也是 m 次单位根

$O(n\log n)$ 的快速傅里叶变换算法

首先有个东西叫做 n 次单位根 ω_n，是一个满足 $\omega_n^n = 1$ 的复数
只不过我们在这里要求，不能存在一个比 n 小的 m 使得 ω_n 也是 m 次单位根

容易发现一个性质 $\frac{1}{n}\sum\limits_{k=0}^{n-1}\omega_n^{vk} = [v \mod n = 0]$

$O(n\log n)$ 的快速傅里叶变换算法

首先有个东西叫做 n 次单位根 ω_n，是一个满足 $\omega_n^n=1$ 的复数只不过我们在这里要求，不能存在一个比 n 小的 m 使得 ω_n 也是 m 次单位根

容易发现一个性质 $\frac{1}{n}\sum_{k=0}^{n-1}\omega_n^{vk}=[v \mod n=0]$

说回多项式乘法，我们知道，多项式乘法的本质是系数序列的卷积，但是这里我们并不直接去做卷积，而是去研究一个更高级的——循环卷积 (如果我们只需要去求普通卷积的话，让 n 大于等于 $|a|+|b|$ 的项数和即可)

$O\left(n\log n\right)$ 的快速傅里叶变换算法

首先有个东西叫做 n 次单位根 ω_n，是一个满足 $\omega_n^n=1$ 的复数
只不过我们在这里要求，不能存在一个比 n 小的 m 使得 ω_n 也是 m 次单位根

容易发现一个性质 $\frac{1}{n}\sum_{k=0}^{n-1}\omega_n^{vk}=[v \mod n=0]$

说回多项式乘法，我们知道，多项式乘法的本质是系数序列的卷积，但是这里我们并不直接去做卷积，而是去研究一个更高级的——循环卷积 (如果我们只需要去求普通卷积的话，让 n 大于等于 $\mid a\mid+\mid b\mid$ 的项数和即可)

$c_i=\sum_{p,q}[p+q \mod n=i]a_pb_q$

$O\left(n\log n\right)$ 的快速傅里叶变换算法

首先有个东西叫做 n 次单位根 ω_n，是一个满足 $\omega_n^n=1$ 的复数只不过我们在这里要求，不能存在一个比 n 小的 m 使得 ω_n 也是 m 次单位根

容易发现一个性质 $\frac{1}{n}\sum\limits_{k=0}^{n-1}\omega_n^{vk}=[v \mod n=0]$

说回多项式乘法，我们知道，多项式乘法的本质是系数序列的卷积，但是这里我们并不直接去做卷积，而是去研究一个更高级的——循环卷积 (如果我们只需要去求普通卷积的话，让 n 大于等于 $|a|+|b|$ 的项数和即可)

$c_i=\sum\limits_{p,q}[p+q \mod n=i]a_pb_q$

联系上一个式子，可以得到

$c_i=\frac{1}{n}\sum\sum\limits_{k=0}^{n-1}\omega_n^{pk+qk-ik}a_pb_q=\frac{1}{n}\sum\limits_{k=0}^{n-1}\omega_n^{-ik}\sum\omega_n^{pk}a_p\sum\omega_n^{qk}b_q$。

$O\left(n\log n\right)$ 的快速傅里叶变换

$O\left(n\log n\right)$ 的快速傅里叶变换

离散傅里叶变换 (Discrete Fourier Transform, DFT) 是要把一个序列 x 变换成另一个序列 X，其中 $X_i = \sum\limits_{k=0}^{n-1} \omega_n^{ik} x_k$。

$O(n\log n)$ 的快速傅里叶变换

离散傅里叶变换 (Discrete Fourier Transform, DFT) 是要把一个序列 x 变换成另一个序列 X，其中 $X_i = \sum\limits_{k=0}^{n-1} \omega_n^{ik} x_k$。

而离散傅里叶逆变换 (Inverse Discrete Fourier Transform, IDFT) 是要把序列 X 变回序列 x，$x_i = \frac{1}{n}\sum\limits_{k=0}^{n-1} \omega_n^{-ik} x_k$。

$O(n\log n)$ 的快速傅里叶变换

离散傅里叶变换 (Discrete Fourier Transform, DFT) 是要把一个序列 x 变换成另一个序列 X，其中 $X_i = \sum_{k=0}^{n-1} \omega_n^{ik} x_k$。

而离散傅里叶逆变换 (Inverse Discrete Fourier Transform, IDFT) 是要把序列 X 变回序列 x，$x_i = \frac{1}{n} \sum_{k=0}^{n-1} \omega_n^{-ik} x_k$。

至于为什么这么就可以变换回来了呢，可以用线性代数去证明，但是我不是很想写，而且容易发现，它究竟是不是逆变换，其实并不重要。

$O\left(n\log n\right)$ 的快速傅里叶变换

离散傅里叶变换 (Discrete Fourier Transform, DFT) 是要把一个序列 x 变换成另一个序列 X，其中 $X_i=\sum\limits_{k=0}^{n-1}\omega_n^{ik}x_k$。
而离散傅里叶逆变换 (Inverse Discrete Fourier Transform, IDFT) 是要把序列 X 变回序列 x，$x_i=\frac{1}{n}\sum\limits_{k=0}^{n-1}\omega_n^{-ik}x_k$。
至于为什么这么就可以变换回来了呢，可以用线性代数去证明，但是我不是很想写，而且容易发现，它究竟是不是逆变换，其实并不重要。
通过上一页最后一行的式子，可以发现，我们求出 a 序列和 b 序列的离散傅里叶变换后，对应位相乘，再进行离散傅里叶逆变换，就可以求出 c 序列。

$O\left(n\log n\right)$ 的快速傅里叶变换

$O\left(n \log n\right)$ 的快速傅里叶变换

如何快速求出这个离散傅里叶变换呢，我们先要介绍两个性质。

$O\left(n \log n\right)$ 的快速傅里叶变换

如何快速求出这个离散傅里叶变换呢，我们先要介绍两个性质。

- $\omega_{2n}^{2k} = \omega_n^k$

$O\left(n\log n\right)$ 的快速傅里叶变换

如何快速求出这个离散傅里叶变换呢，我们先要介绍两个性质。

- $\omega_{2n}^{2k}=\omega_n^k$
- $\omega_{2n}^{k}=-\omega_{2n}^{k+n}$

$O\left(n\log n\right)$ 的快速傅里叶变换

$O(n\log n)$ 的快速傅里叶变换

但是有个很遗憾的事情，就是我们暂时无法快速求出对于任意长度的循环卷积，这里我们要求 n 是 2 的自然数次幂 (只不过当然了，如果只是计算普通卷积的话，我们需要的是一个大于等于 $|a|+|b|$ 的项数和的 n，所以我们让 $n=2^{\lceil\log_2(|a|+|b|)\rceil}$ 就行了)。

$O\left(n\log n\right)$ 的快速傅里叶变换

$O\left(n\log n\right)$ 的快速傅里叶变换

然后接下来我们考虑分治！

$O\left(n\log n\right)$ 的快速傅里叶变换

然后接下来我们考虑分治！

令 $f(x)=\sum_{k=0}^{n} a_i x^i$，$F(x)=\sum_{k=0}^{\frac{n-1}{2}} a_{2i}x^i$，$G(x)=\sum_{k=0}^{\frac{n-1}{2}} a_{2i+1}x^i$(这里的 a_i 就是要变换的序列)。

$O\left(n\log n\right)$ 的快速傅里叶变换

然后接下来我们考虑分治!

令 $f(x)=\sum\limits_{k=0}^{n}a_ix^i$, $F(x)=\sum\limits_{k=0}^{\frac{n-1}{2}}a_{2i}x^i$, $G(x)=\sum\limits_{k=0}^{\frac{n-1}{2}}a_{2i+1}x^i$(这里的 a_i 就是要变换的序列)。

可以注意到 $f(x)=F(x^2)+xG(x^2)$, 而且我们知道 $(\omega_{n+1}^{k})^2=(\omega_{n+1}^{k+\frac{n+1}{2}})^2=\omega_{\frac{n+1}{2}}^{k}$。

$O\left(n\log n\right)$ 的快速傅里叶变换

然后接下来我们考虑分治!

令 $f(x)=\sum\limits_{k=0}^{n} a_i x^i$, $F(x)=\sum\limits_{k=0}^{\frac{n-1}{2}} a_{2i}x^i$, $G(x)=\sum\limits_{k=0}^{\frac{n-1}{2}} a_{2i+1}x^i$(这里的 a_i 就是要变换的序列)。

可以注意到 $f(x)=F(x^2)+xG(x^2)$, 而且我们知道 $(\omega_{n+1}^{k})^2=(\omega_{n+1}^{k+\frac{n+1}{2}})^2=\omega_{\frac{n+1}{2}}^{k}$。

所以我们考虑直接递归下去求 $F(x)$ 和 $G(x)$ 的快速傅里叶变换, 然后可以直接合并出来 $f(x)$ 的快速傅里叶变换, 复杂度明显是 $O(n\log n)$。

$O(n\log n)$ 的快速傅里叶变换

$O\left(n \log n\right)$ 的快速傅里叶变换

但是众所周知递归常数太大了

$O\left(n\log n\right)$ 的快速傅里叶变换

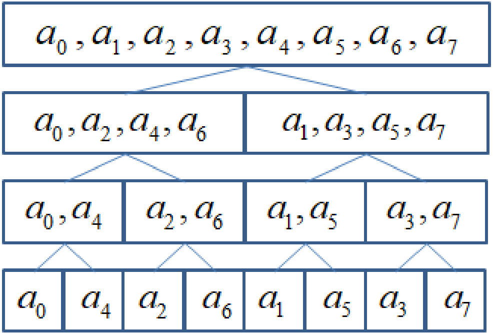
但是众所周知递归常数太大了
然后我们可以发现实际上递归到最后一层时第 i (i 从 0 开始) 个多项式的系数是最开始的多项式的第 $\mathrm{reverse}_i$ 项

$O(n\log n)$ 的快速傅里叶变换

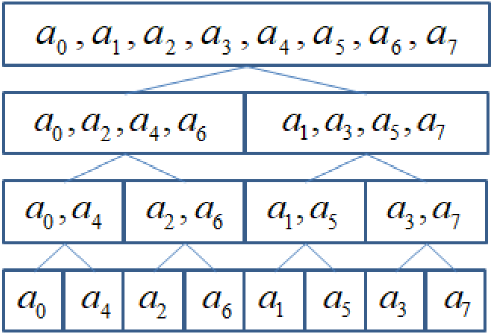
但是众所周知递归常数太大了
然后我们可以发现实际上递归到最后一层时第 i (i 从 0 开始) 个多项式的系数是最开始的多项式的第 reverse_i 项
其中 reverse_i 是 i 在 $\log_2(n+1)$ 位二进制下翻转后的数。

$O(n \log n)$ 的快速傅里叶变换

$O(n\log n)$ 的快速傅里叶变换



$O\left(n\log n\right)$ 的快速傅里叶变换



求出这个后就直接一层层去合并即可，常数变小了不少。

$O(n\log n)$ 的快速傅里叶变换

容易注意到 ω_n^{-1} 也是 n 次单位根，所以用同样的方法去求，然后最后再除以一个 n 即可。

$O(n\log n)$ 的快速数论变换

$O(n \log n)$ 的快速数论变换

假如我们的所有运算都是在 $\bmod p$ 的意义下进行的，而且 p 的原根是 g,

$O(n\log n)$ 的快速数论变换

假如我们的所有运算都是在 $\bmod p$ 的意义下进行的，而且 p 的原根是 g，
容易发现 $g^{\frac{p-1}{n}}$ 在模 p 意义下拥有和 ω_n 一样的性质 (当然，前提是 $n \mid p-1$)

$O(n\log n)$ 的快速数论变换

假如我们的所有运算都是在 $\bmod p$ 的意义下进行的，而且 p 的原根是 g,
容易发现 $g^{\frac{p-1}{n}}$ 在模 p 意义下拥有和 ω_n 一样的性质 (当然，前提是 $n \mid p-1$)
直接替换然后去做就是快速数论变换

$O(n\log n)$ 的快速数论变换

假如我们的所有运算都是在 $\bmod p$ 的意义下进行的，而且 p 的原根是 g，
容易发现 $g^{\frac{p-1}{n}}$ 在模 p 意义下拥有和 ω_n 一样的性质 (当然，前提是 $n \mid p-1$)
直接替换然后去做就是快速数论变换
由于没有浮点数，所以没有精度误差。

$O(n\log n)$ 的快速数论变换

假如我们的所有运算都是在 $\bmod p$ 的意义下进行的，而且 p 的原根是 g,
容易发现 $g^{\frac{p-1}{n}}$ 在模 p 意义下拥有和 ω_n 一样的性质 (当然，前提是 $n \mid p-1$)
直接替换然后去做就是快速数论变换
由于没有浮点数，所以没有精度误差。
一个常见快速数论变换模数 $998244353 = 7 \cdot 17 \cdot 2^{23} + 1$，它的最小的原根是 3。

$O(n\log n)$ 的快速数论变换

假如我们的所有运算都是在 $\bmod p$ 的意义下进行的，而且 p 的原根是 g，
容易发现 $g^{\frac{p-1}{n}}$ 在模 p 意义下拥有和 ω_n 一样的性质 (当然，前提是 $n \mid p-1$)
直接替换然后去做就是快速数论变换
由于没有浮点数，所以没有精度误差。
一个常见快速数论变换模数 $998244353 = 7\cdot 17\cdot 2^{23}+1$，它的最小的原根是 3。
注：应某位同学的要求，补上一个 114514 也是 998244353 的原根。

$O\left(n\log n\right)$ 的 Chirp-Z 变换

$O\left(n\log n\right)$ 的 Chirp-Z 变换

Chirp-Z 变换是可以把离散傅里叶变换中的 ω_n 替换成任意的数 c

$O\left(n \log n\right)$ 的 Chirp-Z 变换

Chirp-Z 变换是可以把离散傅里叶变换中的 ω_n 替换成任意的数 c

而计算 Chirp-Z 变换的 Bluestein 算法可以支持任意的 n，不需要要求 n 是 2 的自然数次幂

$O\left(n \log n\right)$ 的 Chirp-Z 变换

Chirp-Z 变换是可以把离散傅里叶变换中的 ω_n 替换成任意的数 c

而计算 Chirp-Z 变换的 Bluestein 算法可以支持任意的 n，不需要要求 n 是 2 的自然数次幂

考虑 $g_i = \sum\limits_{j=0}^{n-1} f_j \cdot c^{i \cdot j}$

$O\left(n \log n\right)$ 的 Chirp-Z 变换

Chirp-Z 变换是可以把离散傅里叶变换中的 ω_n 替换成任意的数 c

而计算 Chirp-Z 变换的 Bluestein 算法可以支持任意的 n，不需要要求 n 是 2 的自然数次幂

考虑 $g_i = \sum\limits_{j=0}^{n-1} f_j \cdot c^{i \cdot j}$

而且容易发现我们有 $i \cdot j = \binom{i+j}{2} - \binom{i}{2} - \binom{j}{2}$

$O\left(n\log n\right)$ 的 Chirp-Z 变换

Chirp-Z 变换是可以把离散傅里叶变换中的 ω_n 替换成任意的数 c

而计算 Chirp-Z 变换的 Bluestein 算法可以支持任意的 n，不需要要求 n 是 2 的自然数次幂

考虑 $g_i = \sum\limits_{j=0}^{n-1} f_j \cdot c^{i\cdot j}$

而且容易发现我们有 $i \cdot j = \binom{i+j}{2} - \binom{i}{2} - \binom{j}{2}$

于是 $g_i = c^{-\binom{i}{2}} \cdot \sum\limits_{j=0}^{n-1} c^{\binom{i+j}{2}} \cdot c^{-\binom{j}{2}} \cdot f_j$

$O\left(n\log n\right)$ 的 Chirp-Z 变换

$O\left(n\log n\right)$ 的 Chirp-Z 变换

注意到我们可以构造

$O\left(n\log n\right)$ 的 Chirp-Z 变换

注意到我们可以构造

$$A(x)=\sum_{i=0}^{n-1} c^{\binom{i}{2}}\cdot x^i$$

$O(n\log n)$ 的 Chirp-Z 变换

注意到我们可以构造

$$A(x)=\sum_{i=0}^{n-1} c^{\binom{i}{2}}\cdot x^i$$

和

$O(n\log n)$ 的 Chirp-Z 变换

注意到我们可以构造

$$A(x) = \sum_{i=0}^{n-1} c^{\binom{i}{2}} \cdot x^i$$

和

$$B(x) = \sum_{i=0}^{n-1} c^{-\binom{n-i-1}{2}} \cdot f_{n-i-1} \cdot x^i$$

$O\left(n\log n\right)$ 的 Chirp-Z 变换

注意到我们可以构造

$$A(x)=\sum_{i=0}^{n-1}c^{\binom{i}{2}}\cdot x^i$$

和

$$B(x)=\sum_{i=0}^{n-1}c^{-\binom{n-i-1}{2}}\cdot f_{n-i-1}\cdot x^i$$

计算 $C(x)=A(x)\cdot B(x)$

$O\left(n\log n\right)$ 的 Chirp-Z 变换

注意到我们可以构造

$$A(x)=\sum_{i=0}^{n-1}c^{\binom{i}{2}}\cdot x^i$$

和

$$B(x)=\sum_{i=0}^{n-1}c^{-\binom{n-i-1}{2}}\cdot f_{n-i-1}\cdot x^i$$

计算 $C(x)=A(x)\cdot B(x)$

容易发现 $[x^i]C(x)=\sum_{j=0}^{2\cdot n-2}c^{\binom{j}{2}}\cdot c^{-\binom{n+j-i-1}{2}}\cdot f_{n+j-i-1}$

$O\left(n\log n\right)$ 的 Chirp-Z 变换

注意到我们可以构造

$A(x) = \sum\limits_{i=0}^{n-1} c^{\binom{i}{2}} \cdot x^i$

和

$B(x) = \sum\limits_{i=0}^{n-1} c^{-\binom{n-i-1}{2}} \cdot f_{n-i-1} \cdot x^i$

计算 $C(x) = A(x) \cdot B(x)$

容易发现 $[x^i]C(x) = \sum\limits_{j=0}^{2\cdot n-2} c^{\binom{j}{2}} \cdot c^{-\binom{n+j-i-1}{2}} \cdot f_{n+j-i-1}$

于是 $g_i = c^{-\binom{i}{2}} \cdot C_{n+i-1}$

Contents

多项式牛顿迭代法

这个东西的证明我不会，暂时还没时间去学泰勒展开式，但是我们可以把它当作一个结论

多项式牛顿迭代法

这个东西的证明我不会，暂时还没时间去学泰勒展开式，但是我们可以把它当作一个结论

有一个函数 g，我们要求一个 $2 \cdot n - 1$ 次多项式 f 满足 $g(f(x)) \equiv 0 \pmod{x^{2n}}$

多项式牛顿迭代法

这个东西的证明我不会，暂时还没时间去学泰勒展开式，但是我们可以把它当作一个结论
有一个函数 g，我们要求一个 $2 \cdot n - 1$ 次多项式 f 满足 $g(f(x)) \equiv 0 \pmod{x^{2n}}$
首先我们先求一个 $n-1$ 多项式 $f_0(x)$ 满足 $g(f_0(x)) \equiv 0 \pmod{x^n}$

多项式牛顿迭代法

这个东西的证明我不会，暂时还没时间去学泰勒展开式，但是我们可以把它当作一个结论

有一个函数 g，我们要求一个 $2 \cdot n - 1$ 次多项式 f 满足 $g(f(x)) \equiv 0 \pmod{x^{2n}}$

首先我们先求一个 $n-1$ 多项式 $f_0(x)$ 满足 $g(f_0(x)) \equiv 0 \pmod{x^n}$

多项式牛顿迭代法的结论就是 $f(x) \equiv f_0(x) - \frac{g(f_0(x))}{g'(f_0(x))} \pmod{x^{2n}}$

Contents

多项式求逆

多项式求逆

给你一个 $n-1$ 次多项式 $f(x)$，求一个 $n-1$ 次多项式 $g(x)$ 满足 $f(x)\cdot g(x)\equiv 1 \pmod{x^n}$

多项式求逆

给你一个 $n-1$ 次多项式 $f(x)$，求一个 $n-1$ 次多项式 $g(x)$ 满足 $f(x)\cdot g(x)\equiv 1 \pmod{x^n}$

首先为了方便牛顿迭代法，我们先把 n 补到 2 的自然数次幂

多项式求逆

给你一个 $n-1$ 次多项式 $f(x)$，求一个 $n-1$ 次多项式 $g(x)$ 满足 $f(x)\cdot g(x)\equiv 1 \pmod{x^n}$

首先为了方便牛顿迭代法，我们先把 n 补到 2 的自然数次幂

如果 $n=1$，那可以发现 $[x^0]g(x)=([x^0]f(x))^{-1}$

多项式求逆

给你一个 $n-1$ 次多项式 $f(x)$，求一个 $n-1$ 次多项式 $g(x)$ 满足 $f(x) \cdot g(x) \equiv 1 \pmod{x^n}$

首先为了方便牛顿迭代法，我们先把 n 补到 2 的自然数次幂

如果 $n=1$，那可以发现 $[x^0]g(x) = ([x^0]f(x))^{-1}$

不然则构造一个函数 $G(g(x)) = g^{-1}(x) - f(x)$ (其中我们把 $f(x)$ 当作常数)

多项式求逆

给你一个 $n-1$ 次多项式 $f(x)$，求一个 $n-1$ 次多项式 $g(x)$ 满足 $f(x)\cdot g(x)\equiv 1 \pmod{x^n}$

首先为了方便牛顿迭代法，我们先把 n 补到 2 的自然数次幂

如果 $n=1$，那可以发现 $[x^0]g(x)=([x^0]f(x))^{-1}$

不然则构造一个函数 $G(g(x))=g^{-1}(x)-f(x)$ (其中我们把 $f(x)$ 当作常数)

求出一个 g_0 满足 $G(g_0(x))\equiv 0 \pmod{x^{\frac{n}{2}}}$

多项式求逆

给你一个 $n-1$ 次多项式 $f(x)$，求一个 $n-1$ 次多项式 $g(x)$ 满足 $f(x)\cdot g(x)\equiv 1 \pmod{x^n}$

首先为了方便牛顿迭代法，我们先把 n 补到 2 的自然数次幂

如果 $n=1$，那可以发现 $[x^0]g(x)=([x^0]f(x))^{-1}$

不然则构造一个函数 $G(g(x))=g^{-1}(x)-f(x)$ (其中我们把 $f(x)$ 当作常数)

求出一个 g_0 满足 $G(g_0(x))\equiv 0 \pmod{x^{\frac{n}{2}}}$

然后根据多项式牛顿迭代法可以得到

多项式求逆

给你一个 $n-1$ 次多项式 $f(x)$，求一个 $n-1$ 次多项式 $g(x)$ 满足 $f(x)\cdot g(x)\equiv 1 \pmod{x^n}$

首先为了方便牛顿迭代法，我们先把 n 补到 2 的自然数次幂

如果 $n=1$，那可以发现 $[x^0]g(x)=([x^0]f(x))^{-1}$

不然则构造一个函数 $G(g(x))=g^{-1}(x)-f(x)$ (其中我们把 $f(x)$ 当作常数)

求出一个 g_0 满足 $G(g_0(x))\equiv 0 \pmod{x^{\frac{n}{2}}}$

然后根据多项式牛顿迭代法可以得到

$$\begin{aligned} g(x) &\equiv g_0(x)-\frac{g_0^{-1}(x)-f(x)}{-g_0^{-2}(x)} \pmod{x^n} \\ &\equiv 2\cdot g_0(x)+g_0^2(x)\cdot f(x) \pmod{x^n} \end{aligned}$$

多项式求逆

给你一个 $n-1$ 次多项式 $f(x)$，求一个 $n-1$ 次多项式 $g(x)$ 满足 $f(x)\cdot g(x)\equiv 1 \pmod{x^n}$
首先为了方便牛顿迭代法，我们先把 n 补到 2 的自然数次幂
如果 $n=1$，那可以发现 $[x^0]g(x)=([x^0]f(x))^{-1}$
不然则构造一个函数 $G(g(x))=g^{-1}(x)-f(x)$ (其中我们把 $f(x)$ 当作常数)
求出一个 g_0 满足 $G(g_0(x))\equiv 0 \pmod{x^{\frac{n}{2}}}$
然后根据多项式牛顿迭代法可以得到

$$\begin{aligned}g(x)&\equiv g_0(x)-\frac{g_0^{-1}(x)-f(x)}{-g_0^{-2}(x)} \pmod{x^n}\\&\equiv 2\cdot g_0(x)+g_0^2(x)\cdot f(x) \pmod{x^n}\end{aligned}$$

于是我们倍增 + 多项式乘法去做就行了，时间复杂度

Contents

多项式除法和取模

多项式除法和取模

给你一个 n 次多项式 $f(x)$ 和一个 m 次多项式 $g(x)$，求一个 $n-m$ 次多项式 $q(x)$ 和小于 m 次的多项式 $r(x)$ 满足 $g(x) \cdot q(x) + r(x) = f(x)$

多项式除法和取模

给你一个 n 次多项式 $f(x)$ 和一个 m 次多项式 $g(x)$，求一个 $n-m$ 次多项式 $q(x)$ 和小于 m 次的多项式 $r(x)$ 满足
$g(x) \cdot q(x) + r(x) = f(x)$
容易发现如果 $r(x)$ 是 0 就可以直接用多项式求逆去做，所以我们考虑消掉 $r(x)$

多项式除法和取模

给你一个 n 次多项式 $f(x)$ 和一个 m 次多项式 $g(x)$，求一个 $n-m$ 次多项式 $q(x)$ 和小于 m 次的多项式 $r(x)$ 满足

$g(x) \cdot q(x) + r(x) = f(x)$

容易发现如果 $r(x)$ 是 0 就可以直接用多项式求逆去做，所以我们考虑消掉 $r(x)$

定义 $f^R(x) = x^{\deg f(x)} f(x^{-1})$，特别的，我们认为 $\deg r(x) = m - 1$

多项式除法和取模

给你一个 n 次多项式 $f(x)$ 和一个 m 次多项式 $g(x)$，求一个 $n-m$ 次多项式 $q(x)$ 和小于 m 次的多项式 $r(x)$ 满足
$g(x)\cdot q(x)+r(x)=f(x)$
容易发现如果 $r(x)$ 是 0 就可以直接用多项式求逆去做，所以我们考虑消掉 $r(x)$
定义 $f^R(x)=x^{\deg f(x)}f(x^{-1})$，特别的，我们认为 $\deg r(x)=m-1$
容易发现这实际上是将系数翻转过来

多项式除法和取模

多项式除法和取模

而且我们有

多项式除法和取模

而且我们有

$$
\begin{aligned}
f(x^{-1}) &= g(x^{-1}) \cdot q(x^{-1}) + r(x^{-1}) \\
x^n \cdot f(x^{-1}) &= x^m \cdot g(x^{-1}) \cdot x^{n-m} \cdot q(x^{-1}) + x^{n-m+1} \cdot x^{m-1} \cdot r(x^{-1}) \\
f^R(x) &= g^R(x) \cdot q^R(x) + x^{n-m+1} \cdot r^R(x)
\end{aligned}
$$

多项式除法和取模

而且我们有

$$
\begin{aligned}
f(x^{-1}) &= g(x^{-1}) \cdot q(x^{-1}) + r(x^{-1}) \\
x^n \cdot f(x^{-1}) &= x^m \cdot g(x^{-1}) \cdot x^{n-m} \cdot q(x^{-1}) + x^{n-m+1} \cdot x^{m-1} \cdot r(x^{-1}) \\
f^R(x) &= g^R(x) \cdot q^R(x) + x^{n-m+1} \cdot r^R(x)
\end{aligned}
$$

注意到假如我们整个式子对 x^{n-m+1} 取模就可以消掉 $r^R(x)$ 的贡献

多项式除法和取模

而且我们有

$$
\begin{aligned}
f(x^{-1}) &= g(x^{-1}) \cdot q(x^{-1}) + r(x^{-1}) \\
x^n \cdot f(x^{-1}) &= x^m \cdot g(x^{-1}) \cdot x^{n-m} \cdot q(x^{-1}) + x^{n-m+1} \cdot x^{m-1} \cdot r(x^{-1}) \\
f^R(x) &= g^R(x) \cdot q^R(x) + x^{n-m+1} \cdot r^R(x)
\end{aligned}
$$

注意到假如我们整个式子对 x^{n-m+1} 取模就可以消掉 $r^R(x)$ 的贡献

$q^R(x) \equiv \frac{f^R(x)}{g^R(x)} \pmod{x^{n-m+1}}$

多项式除法和取模

而且我们有

$$\begin{aligned} f(x^{-1}) &= g(x^{-1}) \cdot q(x^{-1}) + r(x^{-1}) \\ x^n \cdot f(x^{-1}) &= x^m \cdot g(x^{-1}) \cdot x^{n-m} \cdot q(x^{-1}) + x^{n-m+1} \cdot x^{m-1} \cdot r(x^{-1}) \\ f^R(x) &= g^R(x) \cdot q^R(x) + x^{n-m+1} \cdot r^R(x) \end{aligned}$$

注意到假如我们整个式子对 x^{n-m+1} 取模就可以消掉 $r^R(x)$ 的贡献

$q^R(x) \equiv \frac{f^R(x)}{g^R(x)} \pmod{x^{n-m+1}}$

而且注意到 $q^R(x)$ 只有 $n-m$ 次，所以我们就成功求出了 $q^R(x)$!

多项式除法和取模

而且我们有

$$
\begin{aligned}
f(x^{-1}) &= g(x^{-1}) \cdot q(x^{-1}) + r(x^{-1}) \\
x^n \cdot f(x^{-1}) &= x^m \cdot g(x^{-1}) \cdot x^{n-m} \cdot q(x^{-1}) + x^{n-m+1} \cdot x^{m-1} \cdot r(x^{-1}) \\
f^R(x) &= g^R(x) \cdot q^R(x) + x^{n-m+1} \cdot r^R(x)
\end{aligned}
$$

注意到假如我们整个式子对 x^{n-m+1} 取模就可以消掉 $r^R(x)$ 的贡献
$q^R(x) \equiv \frac{f^R(x)}{g^R(x)} \pmod{x^{n-m+1}}$
而且注意到 $q^R(x)$ 只有 $n-m$ 次，所以我们就成功求出了 $q^R(x)$!
然后系数翻转一下就能得到 $q(x)$，再带进去就能得到 $r(x)$

Contents

多项式开方

多项式开方

给你一个 $n-1$ 次多项式 $f(x)$，求一个 $n-1$ 次多项式 $g(x)$ 满足 $g^2(x) \equiv f(x) \pmod{x^n}$

多项式开方

给你一个 $n-1$ 次多项式 $f(x)$，求一个 $n-1$ 次多项式 $g(x)$ 满足 $g^2(x) \equiv f(x) \pmod{x^n}$
仍然先把 n 补到 2 的自然数次幂

多项式开方

给你一个 $n-1$ 次多项式 $f(x)$，求一个 $n-1$ 次多项式 $g(x)$ 满足 $g^2(x) \equiv f(x) \pmod{x^n}$
仍然先把 n 补到 2 的自然数次幂
如果 $n=1$，可以发现 $[x^0]g(x) = \sqrt{[x^0]f(x)}$

多项式开方

多项式开方

不然则构造一个函数 $G(g(x)) = g^2(x) - f(x)$ (其中我们把 $f(x)$ 当作常数)

多项式开方

不然则构造一个函数 $G(g(x)) = g^2(x) - f(x)$ (其中我们把 $f(x)$ 当作常数)
求出一个 g_0 满足 $G(g_0(x)) \equiv 0 \pmod{x^{\frac{n}{2}}}$

多项式开方

不然则构造一个函数 $G(g(x)) = g^2(x) - f(x)$ (其中我们把 $f(x)$ 当作常数)
求出一个 g_0 满足 $G(g_0(x)) \equiv 0 \pmod{x^{\frac{n}{2}}}$
然后根据多项式牛顿迭代法可以得到

$$
\begin{aligned}
g(x) &\equiv g_0(x) - \frac{g_0^2(x) - f(x)}{2 \cdot g_0(x)} \pmod{x^n} \\
&\equiv \frac{g_0^2(x) + f(x)}{2 \cdot g_0(x)} \pmod{x^n} \\
&\equiv (g_0^2(x) + f(x)) \cdot 2^{-1} \cdot g_0^{-1}(x) \pmod{x^n}
\end{aligned}
$$

多项式开方

不然则构造一个函数 $G(g(x)) = g^2(x) - f(x)$ (其中我们把 $f(x)$ 当作常数)
求出一个 g_0 满足 $G(g_0(x)) \equiv 0 \pmod{x^{\frac{n}{2}}}$
然后根据多项式牛顿迭代法可以得到

$$\begin{aligned} g(x) &\equiv g_0(x) - \frac{g_0^2(x) - f(x)}{2 \cdot g_0(x)} \pmod{x^n} \\ &\equiv \frac{g_0^2(x) + f(x)}{2 \cdot g_0(x)} \pmod{x^n} \\ &\equiv (g_0^2(x) + f(x)) \cdot 2^{-1} \cdot g_0^{-1}(x) \pmod{x^n} \end{aligned}$$

于是我们倍增 + 多项式乘法 + 多项式求逆就行了，时间复杂度 $T(n) = T(\frac{n}{2}) + O(n \log n) = O(n \log n)$。