多项式科技初步

SGColin

目录

1 参考资料

Picks 的博客 Picks's Blog

Miskcoo 的博客 Miskcoo's Space

毛爷爷的论文《再探快速傅里叶变换》

2 多项式

2.1 表示法

次数界：我们称最高次项为 n 次的多项式的次数界为 $n+1$ 。

系数表示 (coefficient representation)：将多项式写作 $A(x)=\sum_{i=0}^{n-1}a_ix^i$，以系数形式表示的，将 n 次多项式 $A(x)$ 的系数 $a_0,a_1,\ldots,a_n$ 看作 $n+1$ 维向量 $(a_0,a_1,\ldots,a_n)$ 的表示方法。

点值表示 (point-value representation)：选取 $n+1$ 个不同的数 $x_0,x_1,\ldots,x_n$ 对多项式进行求值，用集合 $\{(x_i,A(x_i)):0\le i\le n,i\in Z\}$ 来表示多项式的表示方法。

多项式 $A(x)$ 的点值表示不止一种，选取不同的数就可以得到不同的点值表示。但是系数表示和任何一种点值表示都能**唯一确定一个多项式**。

从系数表示转换为点值表示的运算称为**求值**，从点值表示转换为系数表示的运算称为**插值**。

2.2 系数表示法下的基本运算

加法：系数直接相加，复杂度 $\mathcal{O}(n)$。

乘法：也叫**卷积 (convolution)**，两个次数界为 n 的多项式相乘得到次数界为 $2n-1$ 的多项式。暴力复杂度 $\mathrm{O}(n^2)$，使用 FFT 优化到 $\mathrm{O}(n\log n)$。

两个多项式 $A(x)=\sum_{i=0}^{n}a_ix^i$ 和 $B(x)=\sum_{i=0}^{n}b_ix^i$ ，其卷积 $C(x)=A(x)*B(x)$ ，满足

$$C(x)=\sum_{i=0}^{n}c_ix^i\ ,\ c_i=\sum_{k=0}^{i}a_k\times b_{i-k}$$

2.3 点值表示法下的基本运算

此时对于运算相关的多项式有特殊要求，即其点值表示应使用相同的 x 集去计算。

加法：所求值直接相加，复杂度 $\mathrm{O}(n)$。

乘法：所求值直接相乘，复杂度 $\mathrm{O}(n)$，此处复杂度的优越性有着重要意义。

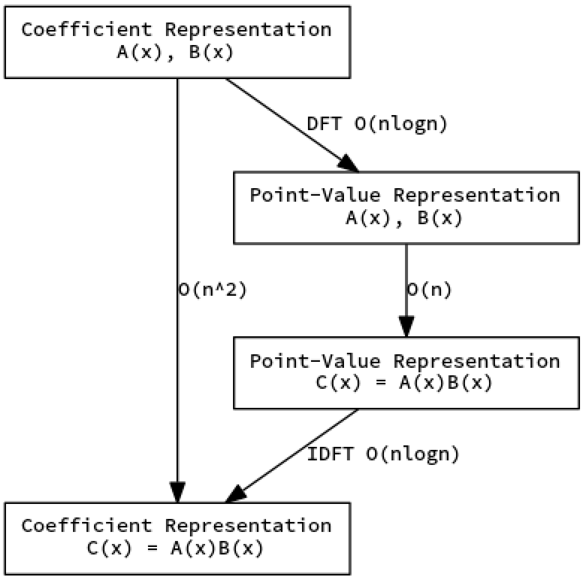
$$A(x)=\{(x_1,y_{(a,1)}),(x_2,y_{(a,2)}),...,(x_{n+1},y_{(a,n+1)})\}\ ,\ B(x)=\{(x_1,y_{(b,1)}),(x_2,y_{(b,2)}),...,(x_{n+1},y_{(b,n+1)})\}$$

$$A(x)\times B(x)=\{(x_1,y_{(a,1)}\times y_{(b,1)}),(x_2,y_{(a,2)}\times y_{(b,2)}),...,(x_{n+1},y_{(a,n+1)}\times y_{(b,n+1)})\}$$

3 Fast Fourier Transform

观察多项式的乘法，可以发现系数表示法下乘法复杂度为 $O(n^2)$，而点值表示法下复杂度为 $O(n)$。如果我们把两个相乘的多项式都先求值出点值表示，然后再在点值表示法下进行乘法，再插值回去，复杂度可能会更优秀。其复杂度瓶颈显然是求值和差值的过程。

快速傅里叶变换 (Fast Fourier Transform) 利用了单位根的性质加速了求值和插值的过程，使得复杂度优化到 $\mathrm{O}(n\log n)$ 。求值的部分称为**离散傅里叶变换 (Discrete Fourier Transform)**，插值的部分称为**逆离散傅里叶变换 (Inverse Discrete Fourier Transform)**。



3.1 复数

复数域常用 $\mathbb{C}$ 表示。

定义 $i^2 = -1$ ，复数写作 $z = a + ib$ 的形式。a 称作实部，b 称作虚部，i 称作虚部单位。

容易看出，复数实际上是一个二元组，因此我们可以把它映射到一个二维平面上，横坐标为实部，纵坐标为虚部。这个平面我们称作复平面。

因此复数的模长 l 即为 $\sqrt{a^2+b^2}$，记作 $|z|$。当点 z 不是原点，即复数 $z \neq 0$ 时，向量与 x 轴正向的夹角称为复数 z 的辐角，记作 $Argz$。

在复平面上复数 z 显然可以表示成 $z = |z|(\cos(Argz) + i\sin(Argz))$ ，称为复数的三角表示。

代数意义下复数的运算

设 $z_1 = a_1 + ib_1$ ，$z_2 = a_2 + ib_2$ ，有：

$z_1 + z_2 = (a_1 + a_2) + i(b_1 + b_2)$

$z_1 \times z_2 = (a_1a_2 - b_1b_2) + i(a_1b_2 + b_1a_2)$

几何意义下复数乘法的性质

现在考虑几何意义下复数的乘法，将两个三角表示的复数相乘。

$$\begin{aligned}
z_1 &= l_1(\cos(\theta) + i\sin(\theta)) \text{ , } l_1 = |z_1| \\
z_2 &= l_2(\cos(\phi) + i\sin(\phi)) \text{ , } l_2 = |z_2| \\
z_1z_2 &= l_1l_2((\cos(\theta)\cos(\phi) - \sin(\theta)\sin(\phi)) + i(\cos(\theta)\sin(\phi) + \sin(\theta)\cos(\phi))) \\
&= l_1l_2(\cos(\theta+\phi) + i\sin(\theta+\phi))
\end{aligned}$$

其中用到了三角函数的两个恒等式。可以发现三角表示下，满足相乘时**模长相乘，幅角相加**。

3.2 单位根

考虑 $z^n = 1$ 的复数解，解有 n 个，第 k 个解可以表示为

$$z_k = \cos\left(\frac{2\pi k}{n}\right) + i\sin\left(\frac{2\pi k}{n}\right)$$

我们称这些复数解为 n 次单位根，显然解一共有 n 个，它们把单位圆等分成 n 个部分。

表示法

我们用 ω 来表示这些单位根，定义 $\omega_n^k = z_k$。

我们称 $\omega_n = \omega_n^1 = z_1 = \cos\left(\frac{2\pi}{n}\right) + i\sin\left(\frac{2\pi}{n}\right)$ 为**主 n 次单位根**。

显然 n 次单位根的模长都为 1，同时相邻两个单位根的夹角相同，我们不难得出单位根的倍数关系 $\omega_n^i = \omega_n * \omega_n^{i-1} = (\omega_n)^i$。因此 n 次单位根组成的数列可以看成公比为 ω_n 的等比数列。

单位根的性质

周期性 $\omega_n^k = \omega_n^{k \bmod n}$ ，因为每一圈都是 n 个单位根。

乘法关系 $\omega_n^j\omega_n^k = \omega_n^{j+k}$ ，根据模长都为 1 和复数乘法的性质可得。

消去引理 $\omega_{dn}^{dk} = \omega_n^k$ ，因为复平面上表示的向量相同。

折半引理 $(\omega_n^k)^2 = (\omega_n^{k+\frac{n}{2}})^2 = \omega_n^{2k}$ $[\ 2 \mid n\ ]$

折半引理正确性是因为当 n 为偶数时单位根是对称的，由此我们也可以得出 $\omega_n^k = -\omega_n^{k+\frac{n}{2}}$。

3.3 计算过程

Discrete Fourier Transform

离散傅里叶变换可以做到 $\mathrm{O}(n\log n)$ 复杂度内求出 **所有 n 次单位根对应的点值**。

为了利用折半引理，我们把多项式补一些系数为 0 的高次项，使得多项式的次数为 2 的幂。

考虑对于一个单位根 ω_n^k ，我们需要求出

$$A(\omega_n^k)=\sum_{i=0}^{n-1}a_j(\omega_n^k)^i$$

我们将奇偶项分离，重新设一个包含偶数项的多项式为 A_0，包含奇数项的多项式为 A_1，有

$$A_0(\omega_n^k)=\sum_{i=0\ \mid\ i\%2=0}^{n-1}a_i(\omega_n^k)^{\frac{i}{2}}\ ,\ A_1(\omega_n^k)=\omega_n^k\sum_{i=0\ \mid\ i\%2=1}^{n-1}a_i(\omega_n^k)^{\frac{i}{2}}$$

由此可以看出多项式 A 可以写作次数界都减半的两个多项式 A_0,A_1 的和

$$A(\omega_n^k)=A_0(\omega_n^{2k})+\omega_n^kA_1(\omega_n^{2k})$$

考虑使用消去引理。我们把 ω_n^k 与 $\omega_n^{k+\frac{n}{2}}$ 对应的答案都写出来

$$\begin{aligned}A(\omega_n^k)&=A_0(\omega_n^{2k})+\omega_n^kA_1(\omega_n^{2k})=A_0(\omega_{\frac{n}{2}}^k)+\omega_n^kA_1(\omega_{\frac{n}{2}}^k)\\A(\omega_n^{k+\frac{n}{2}})&=A_0(\omega_n^{2k})+\omega_n^{k+\frac{n}{2}}A_1(\omega_n^{2k})=A_0(\omega_n^{2k})-\omega_n^kA_1(\omega_n^{2k})=A_0(\omega_{\frac{n}{2}}^k)-\omega_n^kA_1(\omega_{\frac{n}{2}}^k)\end{aligned}$$

注意到如果求出来了 $A_0(\omega_{\frac{n}{2}}^k)$ 和 $A_1(\omega_{\frac{n}{2}}^k)$ 的值，就可以 $\mathrm{O}(1)$ 得到 $A(\omega_n^k)$ 与 $A(\omega_n^{k+\frac{n}{2}})$ 的值。

这是一个分治的形式。也就是说问题变成了两个子问题，问题的规模也缩小了一半都是求 $\frac{n}{2}$ 次多项式，在 $\frac{n}{2}$ 次单位根下的点值。由主定理知

$$T(n)=2T(\frac{n}{2})+\mathrm{O}(n)=\mathrm{O}(n\log n)$$

Inverse Discrete Fourier Transform

设 $y_k=A(\omega_n^k)$ ，考虑另一个多项式 $B(x)=\sum_{k=0}^{n-1}y_kx^k$, 把上面的 n 个单位根的**倒数**代入，得到了另一个离散傅里叶变换 $z_k=B(\omega_n^{-k})$:

$$z_k=\sum_{i=0}^{n-1}y_i(\omega_n^{-k})^i=\sum_{i=0}^{n-1}(\sum_{j=0}^{n-1}a_j(\omega_n^i)^j)(\omega_n^{-k})^i=\sum_{j=0}^{n-1}a_j(\sum_{i=0}^{n-1}(\omega_n^{j-k})^i)$$

此处当 $j=k$ 时，$\sum_{i=0}^{n-1}(\omega_n^{j-k})^i$ 的值等于 n，否则等于 0，证明使用等比数列求和公式即可。

因此有 $z_k=na_k$，即 $a_k=\frac{z_k}{n}$ ，此时我们完成了插值的过程，也就是傅里叶变换的逆变换。

因此我们只需取单位根的倒数作为 x 代入 $B(x)$，用 DFT 优化计算，得到的每个数再除以 n，得到的是 $A(x)$ 的各项系数。

Butterfly Diagram

前人们找到了 DFT 迭代实现的方法，实际上分治最后每个系数的位置是有规律可循的。

考虑在每一次分治，我们都是把二进制末位为 0 的放到左侧，二进制末位为 1 的放到右侧，然后去掉二进制末位。那么最后最靠左的项一定是二进制全部为 0，其次是二进制最高位为 1。

假设 reverse(i) 是将二进制位反转的操作，DFT 分治到最后的系数数组是 B，原来的系数数组是 A，那么有 B[reverse(i)]=A[i]，反转过来之后就是我们刚才描述的分组过程。

蝴蝶变换的本意是优化常数，而逐个求二进制反转的复杂度也是 $O(n \log n)$ 的并不优秀。

这里介绍一种线性递推的方法：$rev[i] = ((rev[i >> 1] >> 1)|(i\&1) << (\log(N) - 1))$。含义是不考虑最高位，剩下的部分反转答案已经在之前求出，然后再单独考虑最高位的答案。

为了实现 swap(A[i],A[reverse(i)]) 并避免同一位置两次交换，我们可以当 $i < reverse(i)$ 时再交换。然后我们就可以模拟分治的过程以优化常数了。

```
for (int i = 1; i < l; ++i)
  rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit - 1));
for (int i = 0; i < l; ++i) if (rev[i] > i) swap(a[i], a[rev[i]]);
```

4 Fast Number-Theoretic Transform

4.1 原根

快速数论变换提出，从数论角度找出一个具有单位根性质的东西。

定义**素数** p **的原根** g 为使得 $g^0, g^1, \dots, g^{p-2} \pmod p$ 互不相同的数。

主单位根的选取与循环性

如果我们取素数 $p = k \times 2^n + 1$，找到它的原根 g，那么主单位根就是 $g_n \equiv g^k \pmod p$，其幂就相当于更改 $k \times a$ 中 a 的值，显然是两两不同的，因此 $(g_n)^0, (g_n)^1, \dots, (g_n)^{p-2} \pmod p$ 也就满足了互不相同的性质。同时也就有了特殊值 $g_n^0 \equiv g_n^n = g^{nk} \equiv 1 \pmod p$ 的性质。

折半引理在数论意义下的体现

由于 p 是素数，并且 $g_n^n \equiv 1 \pmod p$，这样 $g_n^{\frac{n}{2}} \pmod p$ 必然是 -1 或 1，再根据 g_k 互不相同这个特点，所以 $g_n^{\frac{n}{2}} \equiv -1 \pmod p$，满足折半引理。

4.2 计算过程

因此我们可以用原根替代单位根，运算变为模意义下，代码与 FFT 基本相同。更多的形如 $k \times 2^n + 1$ 模数与对应的原根可以去 这里 看。

5 多项式乘法

5.1 问题描述

给定两个多项式 $A(x)$ 和 $B(x)$

$$A(x)=\sum_{i=0}^{n}a_ix^i\ ,\ B(x)=\sum_{i=0}^{n}b_ix^i$$

求卷积 $C(x)=A(x)*B(x)$ ，满足

$$C(x)=\sum_{i=0}^{n}c_ix^i\ ,\ c_i=\sum_{k=0}^{i}a_k\times b_{i-k}$$

5.2 解决方法

见上述 Fast Fourier Transform 与 Fast Number-Theoretic Transform 的内容。

5.3 代码实现

使用 [UOJ 34] 多项式乘法 作为测试题。

Fast Fourier Transform，使用的是 3 次变换的最基本写法，用时 362 ms

```
inline void FFT(Complex *f, int len, int o) {
    for (int i = 0; i < len; ++i)
      if (rev[i] > i) swap(f[i], f[rev[i]]);
    for (int i = 1; i < len; i <<= 1) {
      Complex wn = Complex(cos(PI / i) , o * sin(PI / i));
      for (int j = 0; j < len; j += (i << 1)) {
        Complex w = Complex(1, 0), x, y;
        for (int k = 0; k < i; ++k, w = w * wn) {
          x = f[j + k]; y = w * f[i + j + k];
          f[i + j + k] = x - y; f[j + k] = x + y;
        }
      }
    }
    if (o == -1) for (int i = 0; i < len; ++i) f[i].x /= len;
}
```

Fast Number-Theoretic Transform，模数为 998244353，用时 403 ms

```cpp
inline void NTT(int *f, int len, int o) {
  for (int i = 1; i < len; ++i)
    if (i > rev[i]) swap(f[i], f[rev[i]]);
  for (int i = 1; i < len; i <<= 1) {
    int wn = qpow(3, (mod - 1) / (i << 1));
    if (o == -1) wn = qpow(wn, mod - 2);
    for (int j = 0; j < len; j += (i << 1)) {
      int w = 1, x, y;
      for (int k = 0; k < i; ++k, w = 1ll * w * wn % mod) {
        x = f[j + k]; y = 1ll * w * f[i + j + k] % mod;
        f[j + k] = mo(x + y); f[i + j + k] = mo(x - y + mod);
      }
    }
  }
  if (o == -1) {
    int invl = qpow(len, mod - 2);
    for (int i = 0; i < len; ++i) f[i] = 1ll * f[i] * invl % mod;
  }
}
```

6 多项式求逆

6.1 问题描述

给定一个 n 次多项式 $A(x)$ ，求出一个多项式 $B(x)$，满足

$$A(x) * B(x) \equiv 1 \pmod{x^{n+1}}$$

系数对 998244353 取模。

6.2 解决方法

采用倍增的思想。

考虑只有常数项的时候，$A(x) \equiv c \pmod x$ ，那么 $A^{-1}(x)$ 即为 c^{-1}。

对于 $n > 1$ 的时候，设 $B(x) = A^{-1}(x)$ ，有

$$A(x)B(x) \equiv 1 \pmod{x^n}$$

因为此时模 x^n 相当于只保留多项式前 n 项，所以该同余式在模 $x^k, 0 \le k \le n$ 时都成立

$$A(x)B(x) \equiv 1 \pmod{x^{\lceil \frac{n}{2} \rceil}}$$

假设在 $\pmod{x^{\lceil \frac{n}{2} \rceil}}$ 意义下 $A(x)$ 的逆元是 $B\ (x)$ 并且我们已经求出，那么

$$A(x)B'(x) \equiv 1 \pmod{x^{\lceil \frac{n}{2} \rceil}}$$

两式相减，得

$$B(x) - B'(x) \equiv 0 \pmod{x^{\lceil \frac{n}{2} \rceil}}$$

两边平方，得

$$B^2(x) - 2B(x)B'(x) + B'^2(x) \equiv 0 \pmod{x^n}$$

模数平方的合法性在于，次数大于 n 的系数，对应的卷积中每组乘法必然有一项为 0。

两侧同乘 $A(x)$ ，整理得

$$B(x) \equiv 2B'(x) - A(x)B'^2(x) \pmod{x^n}$$

因此只需将 $B'(x)$ 和 $A(x)$ 在模 x^n 意义下的插值求出，有

$$B_i = 2B'_i - A_iB'^2_i = B'_i(2 - A_iB')$$

因此一遍 NTT 就可以由 $B'(x)$ 求出 $B(x)$ 了。

总的时间复杂度为

$$T(n) = T(\frac{n}{2}) + \mathcal{O}(n \log n) = \mathcal{O}(n \log n)$$

由此过程也可以得到一个结论：**一个多项式有没有逆元完全取决于其常数项是否有逆元。**

6.3 代码实现

使用 [Luogu P4238] 多项式求逆 作为测试题。

递归版本，使用 O2 优化，用时 562 ms

```
inline void Inv(int *a, int *b, int n) {
  if (n == 1) {b[0] = qpow(a[0], mod - 2); return;}
  Inv(a, b, (n + 1) >> 1);
  int len = Rev(n << 1);
  for (int i = 0; i < n; ++i) tmp[i] = a[i];
  for (int i = n; i < len; ++i) b[i] = tmp[i] = 0;
  NTT(b, len, 1); NTT(tmp, len, 1);
  for (int i = 0; i < len; ++i)
    b[i] = (2ll - 1ll * tmp[i] * b[i] % mod + mod) * b[i] % mod;
  NTT(b, len, -1);
  for (int i = 0; i < len; ++i) tmp[i] = 0;
  for (int i = n; i < len; ++i) b[i] = 0;
}
```

迭代版本，使用 O2 优化，用时 570 ms

```
inline void Inv(int *a, int *b, int n) {
  b[0] = qpow(a[0], mod - 2);
  int len;
  for (int l = 1; l < (n << 1); l <<= 1) {
    len = Rev(l << 1);
    for (int i = 0; i < l; ++i) tmp[i] = a[i];
    NTT(b, len, 1); NTT(tmp, len, 1);
    for (int i = 0; i < len; ++i)
      b[i] = (2ll - 1ll * tmp[i] * b[i] % mod + mod) * b[i] % mod;
    NTT(b, len, -1);
    for (int i = l; i < len; ++i) b[i] = 0;
  }
}
```

7 多项式开根

7.1 问题描述

给定一个 n 次多项式 $A(x)$，求一个在 $\bmod x^{n+1}$ 意义下的多项式 $B(x)$，使得

$$B^2(x) \equiv A(x) \pmod{x^{n+1}}$$

多项式的系数在模 998244353 意义下进行运算，保证常数项 $a_0 = 1$。

7.2 解决方法

同样采用倍增的思想。

如果只有常数项，$A(x) \equiv c \pmod x$ ，那么 $\sqrt{A(x)}$ 即为 $\sqrt{c} \equiv 1 \pmod x$ （二次剩余）。

对于 $n > 1$ 的时候，同样根据上一题的结论，我们可以把问题范围缩小到 $\bmod x^{\lceil \frac{n}{2} \rceil}$ ，有

$$B^2(x) \equiv A(x) \pmod{x^n} \Rightarrow B^2(x) \equiv A(x) \pmod{x^{\lceil \frac{n}{2} \rceil}}$$

不妨设我们已经求出来了 $\bmod x^{\lceil \frac{n}{2} \rceil}$ 意义下的根 $D(x)$ ，即

$$D^2(x) \equiv A(x) \pmod{x^{\lceil \frac{n}{2} \rceil}}$$

因此 $B(x)$ 与 $D(x)$ 在模 $x^{\lceil \frac{n}{2} \rceil}$ 意义下同余，移项得

$$B(x) - D(x) \equiv 0 \pmod{x^{\lceil \frac{n}{2} \rceil}}$$

两侧平方，得

$$B^2(x) + D^2(x) - 2B(x)D(x) \equiv 0 \pmod{x^n}$$

模数能平方的原因与上一题相同。

我们知道 $\bmod x^n$ 时 $B^2(x)$ 即为 $A(x)$ ，因此

$$A(x) + D^2(x) - 2B(x)D(x) \equiv 0 \pmod{x^n}$$

移项，得

$$B(x) \equiv \frac{D^2(x) + A(x)}{2D(x)} \equiv \left(D(x) + \frac{A(x)}{D(x)}\right) \times 2^{-1} \pmod{x^n}$$

因此倍增时进行多项式求逆即可，总的时间复杂度为

$$T(n) = T(\frac{n}{2}) + \mathcal{O}(n \log n) = \mathcal{O}(n \log n)$$

7.3 代码实现

使用 [Luogu P5205] 多项式开根 作为测试题。

递归版本，使用 O2 优化，用时 3081 ms

```
inline void Sqrt(int *a, int *b, int n) {
  if (n == 1) {b[0] = 1; return;}
  Sqrt(a, b, (n + 1) >> 1);
  Inv(b, b0, n);
  int len = Rev(n << 1);
  for (int i = 0; i < n; ++i) a0[i] = a[i];
  for (int i = n; i < len; ++i) a0[i] = 0;
  NTT(a0, len, 1); NTT(b0, len, 1);
  for (int i = 0; i < len; ++i) a0[i] = 1ll * a0[i] * b0[i] % mod;
  NTT(a0, len, -1);
  for (int i = 0; i < n; ++i)
    b[i] = 1ll * (b[i] + a0[i]) % mod * inv2 % mod;
  for (int i = n; i < len; ++i) b[i] = 0;
}
```

迭代版本，使用 O2 优化，用时 3104 ms

```
inline void Sqrt(int *a, int *b, int n) {
  b[0] = 1;
  int len;
  for (int l = 1; l < (n << 1); l <<= 1) {
    Inv(b, b0, l);
    len = Rev(l << 1);
    for (int i = 0; i < l; ++i) a0[i] = a[i];
    for (int i = l; i < len; ++i) a0[i] = 0;
    NTT(a0, len, 1); NTT(b0, len, 1);
    for (int i = 0; i < len; ++i) a0[i] = 1ll * a0[i] * b0[i] % mod;
    NTT(a0, len, -1);
    for (int i = 0; i < l; ++i)
      b[i] = 1ll * (b[i] + a0[i]) % mod * inv2 % mod;
    for (int i = l; i < len; ++i) b[i] = 0;
  }
}
```

8 多项式除法和取模

8.1 问题描述

给定一个 n 次多项式 $A(x)$ 和一个 m 次多项式 $B(x)$，求出多项式 $D(x), R(x)$，满足

$$A(x) = D(x)B(x) + R(x)$$

$D(x)$ 次数为 $n-m$，$R(x)$ 次数小于 m ，所有的运算在模 998244353 意义下进行。

8.2 解决方法

注意到带着 $R(x)$ 在这里很麻烦，前人们想到了一个神奇的解决办法。

设 $A^R(x) = x^n A(\frac{1}{x})$ ，我们将右侧展开：

$$A^R(x) = x^n A(\frac{1}{x}) = x^n \sum_{i=0}^{n} a_i \frac{1}{x^i} = \sum_{i=0}^{n} a_i x^{n-i}$$

所以 $A^R(x)$ 就是将 $A(x)$ 的**系数反转**。

我们将所求的等式中 x 全部换成 $\frac{1}{x}$ ，然后两侧同乘 x^n ：

$$x^n A(\frac{1}{x}) = x^{n-m} D(\frac{1}{x}) x^m B(\frac{1}{x}) + x^{n-m+1} x^{m-1} R(\frac{1}{x})$$

$$A^R(x) = D^R(x) B^R(x) + x^{n-m+1} R^R(x)$$

注意到 $D^R(x)$ 最高次反转后不变，依然为 $n-m$。

而右侧的 $R^R(x)$ 因为前面有 x^{n-m+1} 所以最低次为 $n-m+1$ 。

所以我们可以把多项式运算在 $\text{mod} x^{n-m+1}$ 意义下进行，这样 $R(x)$ 就消失了：

$$A^R(x) \equiv D^R(x) B^R(x) \pmod{x^{n-m+1}}$$

因此就可以得到 $D^R(x)$ 的解法：

$$\frac{A^R(x)}{B^R(x)} \equiv D^R(x) \pmod{x^{n-m+1}}$$

一个多项式求逆就可以求出 $D^R(x)$ 了，再将 $D^R(x)$ 进行反转就得到了答案。

将求出的 $D(x)$ 回代，再进行一次减法即可求出 $R(x)$。

复杂度与多项式求逆同阶，为 $\mathcal{O}(n \log n)$ 。

8.3 代码实现

使用 [Luogu P4512] 多项式除法 作为测试题。

多项式求逆递部分归版本，使用 O2 优化，用时 710 ms

注意求 $D(x)$ 部分的那次卷积是在 $\bmod x^{n-m+1}$ 意义下的，所以 $A^R(x)$ 和 $B^R(x)$ 中，次数高于 $n-m+1$ 的项需要清空（因为 FFT 卷积过程中高次系数也会对低次系数造成影响）

```
inline void Div(int *a, int *b, int n, int m) {
  for (int i = 0; i <= n; ++i) ar[i] = a[n - i];
  for (int i = 0; i <= m; ++i) br[i] = b[m - i];
  for (int i = n - m + 2; i <= n; ++i) ar[i] = 0;
  for (int i = n - m + 2; i <= m; ++i) br[i] = 0;
  Inv(br, invb, n - m + 1);
  int len = Rev((n - m + 1) << 1);
  NTT(ar, len, 1); NTT(invb, len, 1);
  for (int i = 0; i < len; ++i) ar[i] = 1ll * ar[i] * invb[i] % mod;
  NTT(ar, len, -1);
  for (int i = 0; i <= n - m; ++i) tmp[i] = d[i] = ar[n - m - i];
  len = Rev(n << 1);
  for (int i = n - m + 1; i <= len; ++i) tmp[i] = 0;
  NTT(b, len, 1); NTT(tmp, len, 1);
  for (int i = 0; i < len; ++i) b[i] = 1ll * b[i] * tmp[i] % mod;
  NTT(b, len, -1);
  for (int i = 0; i <= m; ++i) r[i] = mo(a[i] - b[i] + mod);
}
```

9 分治 FFT

9.1 问题描述

给定长度为 $n-1$ 的数组 $g[1],...,g[n-1]$，求长度为 n 的数组 $f[0],...,f[n-1]$，其中

$$f[i]=\sum_{j=1}^{i}f[i-j]g[j]$$

边界为 $f[0]=1$，运算在模 998244353 下进行。

9.2 解决方法

分治求解

暴力做是 $\mathcal{O}(n^2)$ 的，考虑使用类似 CDQ 分治的思想，每次我们求出 $[L,mid]$ 范围内的 f 数组之后，把这部分 f 对 $[mid+1,R]$ 范围内 f 的贡献一起做。

考虑对 $x\in[mid+1,R]$ 的 $f[x]$ 的贡献 w_x，有

$$w_x=\sum_{i=l}^{mid}f[i]g[x-i]$$

因此卷积求 w 数组，注意求 w_x 时后半段的 f 认为是 0，否则会存在右区间内部贡献的清空。

总的时间复杂度为

$$T(n)=2T(\frac{n}{2})+\mathcal{O}(n\log n)=\mathcal{O}(n\log^2 n)$$

多项式求逆

一阶分治 FFT 是可以看作卷积处理的。

不妨设将数组看成多项式，有

$$F(x)=\sum_{i=0}^{n-1}f[i]x^i\ ,\ G(x)=\sum_{i=0}^{n-1}g[i]x^i$$

将两个多项式卷积，有

$$F(x)G(x)=\sum_{i=0}^{\infty}x^i\sum_{j}f[i-j]g[j]=F(x)-f[0]$$

后一个等式成立的原因是，注意到后一个求和就是 $f[i]$ 的形式，所以只有 $f[0]$ 没有被计数

求 f 数组可以看作是 $\text{mod}x^n$ 意义下进行的，因此有

$$F(x)G(x)\equiv F(x)-f[0]\pmod x^n\ \Rightarrow\ F(x)\equiv\frac{f_0}{1-G(x)}\equiv(1-G(x))^{-1}\pmod x^n$$

于是一遍多项式求逆就可以求出来了，复杂度为 $\mathcal{O}(n\log n)$

9.3 代码实现

使用 [Luogu P4721] 分治 FFT 作为测试题。

分治版本，使用 O2 优化，用时 919 ms

```
void solve(int l, int r, int bit) {
  if (bit <= 0) return;
  solve(l, mid, bit - 1);
  int len = Rev(r - l);
  for (int i = 0; i < len / 2; ++i) a[i] = f[l + i];
  for (int i = len / 2; i < len; ++i) a[i] = 0;
  for (int i = 0; i < len; ++i) b[i] = g[i];
  NTT(a, len, 1); NTT(b, len, 1);
  for (int i = 0; i < len; ++i) a[i] = 1ll * a[i] * b[i] % mod;
  NTT(a, len, -1);
  for (int i = len / 2; i < len; ++i) f[l + i] = mo(f[l + i] + a[i]);
  solve(mid, r, bit - 1);
}
```

多项式求逆版本，使用 O2 优化，用时 261 ms

```
inline void solve(int *a, int n) {
  a[0] = 1;
  for (int i = 1; i < n; ++i) a[i] = mo(mod - a[i]);
  Inv(a, b, n);
  for (int i = 0; i < n; ++i) print(b[i], 0);
}
```

10 多项式求导和积分

10.1 问题描述

给定一个 n 次多项式 $A(x)$

求一个 $n-1$ 次多项式 $B(x)$ ，和一个 $n+1$ 次多项式 $C(x)$，满足

$$B(x)=A'(x)\ ,\ C(x)=\int A(x)$$

10.2 解决方法

直接按照定义做即可，有

$$B(x)=\sum_{i=1}^{n} ia_ix^{i-1}\ ,\ C(x)=\sum_{i=1}^{n}\frac{a_ix^{i+1}}{i+1}$$

我们一般认为 $C(x)$ 的常数项为 0。

复杂度显然为 $\mathcal{O}(n)$。

10.3 代码实现

```
inline void Der(int *a, int n) {
  for (int i = 1; i < n; ++i) a[i - 1] = 1ll * i * a[i] % mod;
  a[n - 1] = 0;
}

inline void Int(int *a, int n) {
  for (int i = n; i; --i) a[i] = 1ll * a[i - 1] * qpow(i, mod - 2) % mod;
  a[0] = 0;
}
```

11 多项式对数函数

11.1 问题描述

给出 n 次多项式 $A(x)$，求一个 $\bmod x^{n+1}$ 下的多项式 $B(x)$，满足

$$B(x) \equiv \ln A(x) \pmod{x^{n+1}}$$

11.2 解决方法

设 $F(x) = \ln x$，则 $B(x) = F(A(x))$ 。

对 $B(x)$ 求导，根据链式法则，有

$$B'(x) = F'(A(x))A'(x) = \frac{A'(x)}{A(x)}$$

因此对 $A(x)$ 分别进行求导和求逆，卷积即可求出 $B'(x)$ ，再对其进行积分即可。

复杂度与多项式求逆同阶，为 $\mathcal{O}(n \log n)$ 。

11.3 代码实现

使用 [Luogu P4725] 多项式对数函数 作为测试题，不再区分多项式求逆部分的实现方式。

多项式求逆递部分归版本，使用 O2 优化，用时 682 ms

```
inline void Ln(int *a, int *b, int n) {
  Inv(a, b, n); Der(a, n);
  int len = Rev(n << 1);
  NTT(a, len, 1); NTT(b, len, 1);
  for (int i = 0; i < len; ++i) a[i] = 1ll * a[i] * b[i] % mod;
  NTT(a, len, -1); Int(a, n);
}
```