1.6 多项式

拉格朗日插值

- 已知最高次数不超过 $n-1$ 的多项式在平面上的 n 个点 $(x_1,y_1),(x_2,y_2),\dots,(x_n,y_n)$，满足 $\forall 1\le i<j\le n, x_i\ne x_j$，则可还原出多项式：

$$f(x)=\sum_{i=1}^{n}y_i\prod_{j\ne i}\frac{x-x_j}{x_i-x_j}$$

- 若 $x_i=i$，且 $x>n$，则该式可简化为：

$$f(x)=\left(\prod_{i=1}^{n}(x-i)\right)\left(\sum_{i=1}^{n}\frac{(-1)^{n-i}y_i}{(x-i)(i-1)!(n-i)!}\right)$$

- 预处理相关逆元和前缀积，即可 $\mathcal{O}(n)$ 计算。

- 若已知二元多项式在空间内的 nm 个点 $(x_{11}, y_{11}, z_{11}), \dots, (x_{nm}, y_{nm}, z_{nm})$，满足任意两点横纵坐标至少一个不相等，且该多项式中 x 的最高次数不超过 $n-1$，y 的最高次数不超过 $m-1$，则可还原出多项式：

$$f(x,y) = \sum_{i=1}^{n}\sum_{j=1}^{m} z_{ij} \prod_{k\neq i} \frac{x - x_{kj}}{x_{ij} - x_{kj}} \prod_{l\neq j} \frac{x - x_{il}}{x_{ij} - x_{il}}$$

FFT

- 即快速傅里叶变换。
- 设 n 次单位根 $\omega_n^k = \cos\frac{2\pi k}{n} + i\sin\frac{2\pi k}{n}$。
- 将多项式 A, B 的项数补到 2 的整数次幂，设项数为 n，采用分治法快速求出 A, B 代入 $\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1}$ 的点值，将点值相乘再通过类似的过程还原回多项式，即可快速求出多项式 $A \times B$，时间复杂度 $\mathcal{O}(n\log n)$。

DFT

- 即离散傅里叶变换。
- 对于多项式

$$A(x) = \sum_{k=0}^{n-1} a_k x^k = \sum_{k=0}^{\frac{n}{2}-1} (a_{2k}x^{2k} + a_{2k+1}x^{2k+1}) = \sum_{k=0}^{\frac{n}{2}-1} a_{2k}(x^2)^k + x\sum_{k=0}^{\frac{n}{2}-1} a_{2k+1}(x^2)^k$$

- 设 $A_1(x) = \sum\limits_{k=0}^{\frac{n}{2}-1} a_{2k}x^k, A_2(x) = \sum\limits_{i=0}^{\frac{n}{2}-1} a_{2k+1}x^k$，若已求得 $A_1(\omega_{\frac{n}{2}}^k), A_2(\omega_{\frac{n}{2}}^k), 0 \le k \le \frac{n}{2} - 1$，则

$$\begin{aligned} A(\omega_n^k) &= A_1(\omega_n^{2k}) + w_n^k A_2(w_n^{2k}) = A_1(\omega_{\frac{n}{2}}^k) + w_n^k A_2(\omega_{\frac{n}{2}}^k) \\ A(\omega_n^{k+\frac{n}{2}}) &= A_1(\omega_n^{2k}) - w_n^{k+\frac{n}{2}} A_2(w_n^{2k}) = A_1(\omega_{\frac{n}{2}}^k) - w_n^k A_2(\omega_{\frac{n}{2}}^k) \end{aligned}$$

- 不断分治下去，此时我们便求得了 $A(\omega_n^k), 0 \le k \le n-1$。

IDFT

- 即逆离散傅里叶变换，由之前的叙述我们有：

$$\begin{bmatrix} (\omega_n^0)^0 & (\omega_n^0)^1 & \cdots & (\omega_n^0)^{n-1} \\ (\omega_n^1)^0 & (\omega_n^1)^1 & \cdots & (\omega_n^1)^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ (\omega_n^{n-1})^0 & (\omega_n^{n-1})^1 & \cdots & (\omega_n^{n-1})^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} A(\omega_n^0) \\ A(\omega_n^1) \\ \vdots \\ A(\omega_n^{n-1}) \end{bmatrix}$$

- 对矩阵求逆，可得：

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \frac{1}{n} \begin{bmatrix} (\omega_n^{-0})^0 & (\omega_n^{-0})^1 & \cdots & (\omega_n^{-0})^{n-1} \\ (\omega_n^{-1}))^0 & (\omega_n^{-1})^1 & \cdots & (\omega_n^{-1})^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ (\omega_n^{-(n-1)})^0 & (\omega_n^{-(n-1)})^1 & \cdots & (\omega_n^{-(n-1)})^{n-1} \end{bmatrix} \begin{bmatrix} A(\omega_n^0) \\ A(\omega_n^1) \\ \vdots \\ A(\omega_n^{n-1}) \end{bmatrix}$$

- 不难发现，将 ω_n^k 替换成 ω_n^{-k} 再做一遍 **DFT**，最后将结果除以 n 即可实现 **IDFT**。

位逆序变换

- 为了减小递归实现带来的常数，我们考虑直接得到递归最底层的排列顺序，再逐层向上合并。
- 不难证明，设 $n = 2^m$，则系数 a_i 递归到最底层时的下标恰好为 i 在 m 位二进制表示下的对称翻转，我们称这个变换为**位逆序变换（蝴蝶变换）**。
- 设 $rev[i]$ 表示系数 a_i 递归到最底层时的下标，则显然有递推式

```
    int m = 0, _n = na + nb; //相乘的两个多项式最高次数分别为 na,nb
    for (n = 1; n <= _n; n <<= 1)
        ++m;
    for (int i = 1; i < n; ++i)
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (m - 1));
```

- 最终我们得到了 **FFT** 的迭代实现。

```
typedef long double ld;
typedef complex<ld> com;
const ld pi = acos(-1.0);

inline void DFT(vector<com> &a, int opt)
{
    int n = a.size();
    for (int i = 0; i < n; ++i)
        if (i < rev[i])
            std::swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k <<= 1)
    {
        com w(cos(pi / k), opt * sin(pi / k));;
        for (int i = 0; i < n; i += k << 1)
        {
            com res(1.0, 0.0);
            for (int j = 0; j < k; ++j)
            {
                com u = a[i + j],
                    v = res * a[i + j + k];
                a[i + j] = u + v;
                a[i + j + k] = u - v;
                res = res * w;
            }
        }
    }
    if (opt == -1)
    {
        for (int i = 0; i < n; ++i)
            a[i] /= n;
    }
}
```

NTT

- 即快速数论变换。
- 在模质数 P 意义下，原根 g 具有单位根的性质：

$$(g^k)^{P-1} \equiv 1(\mod P), g^{\frac{P-1}{2}} \equiv -1(\mod P)$$

- 若 $P=2^x a+1$，设 $n=2^m$，用 $g^{2^{x-m}a}$ 替换 ω_n 即可实现 **NTT**。
- 常见的 P 有：

$$P=1004535809=479\times 2^{21}+1,g=3$$
$$P=998244353=7\times 17\times 2^{23}+1,g=3$$

- 实现时可以预处理 $g^{2^{x-m}a}$ 的幂次，减小常数，代码见多项式基本操作。

多项式基本操作

- 相关数组应开到题目给定长度的四倍，线性逆元应预处理到题目给定长度的两倍。

快速卷积的变式

- 欲求 $h_k=\sum\limits_{i=0}^{n-k} f_i g_{i+k}$，令 $g'_{n-i-k}=g_{i+k},h'_{n-k}=h_k$，则原式可变为：

$$h'_{n-k}=\sum_{i=0}^{n-k} f_i g'_{n-k-i}$$

- 做快速卷积即可。

分治NTT

- 以计算 $f_i=\sum\limits_{j=1}^{i} f_{i-j}g_j(1\le i<n)$ 为例，$g_1,g_2,\dots,g_{n-1}$ 已知，$f_0=1$。
- 对于当前区间 $[l,r]$，取中点 mid。
  - 先递归区间 $[l,mid]$。
  - 做 $g_1,\dots,g_{r-l}$ 和 $f_l,f_{l+1},\dots,f_{mid}$ 的卷积，求出两者对 $f_{mid+1},f_{mid+2},\dots,f_r$ 的贡献。
  - 再递归区间 $[mid+1,r]$。
- 时间复杂度 $\mathcal{O}(n\log^2 n)$。

牛顿迭代

- 当前有一个函数 g，要求解出一个多项式 f 的前 n 项，使得 $g(f)=0$。
- 已知 f 的前 n 项 f_0，现在想求出 f 的前 $2n$ 项，由泰勒展开：

$$0\equiv g(f_0)+g'(f_0)(f-f_0)\pmod{x^{2n}}$$
$$f\equiv f_0-\frac{g(f_0)}{g'(f_0)}\pmod{x^{2n}}$$

多项式求逆

- 考虑和牛顿迭代类似的倍增法。
- 设当前求出 $B_0(x)$ 使得 $A(x)B_0(x)\equiv 1\pmod{x^n}$，则

$$[A(x)B_0(x)-1]^2\equiv 0\pmod{x^{2n}}$$
$$A(x)[2B_0(x)-A(x)B_0^2(x)]\equiv 1\pmod{x^{2n}}$$

- 不断迭代下去即可，时间复杂度 $\mathcal{O}(n\log n)$。

多项式取对数

- 对 $\ln(A(x))$ 求导后再积分回来，则 $B(x)=\int\frac{A'(x)}{A(x)}\mathrm{d}x$。
- 需保证 $A(x)$ 的常数项为 1。

多项式求指数

- 套用牛顿迭代，令 $g(x)=\ln(x)-A$，最终要使 $g(B)=0$，代入上式，有

$$B(x)\equiv B_0(x)[1-\ln B_0(x)+A(x)]\pmod{x^{2n}}$$

- 需保证 $A(x)$ 的常数项为 0。

多项式开根

- 套用牛顿迭代，令 $g(x)=x^2-A$，最终要使 $g(B)=0$，代入上式，有

$$B(x)\equiv\frac{B_0(x)^2+A(x)}{2B_0(x)}\pmod{x^{2n}}$$

- 若 $A(x)$ 的常数项不为 1，初始时求 B 的常数项需求二次剩余。

多项式快速幂

- 朴素的算法即直接倍增，时间复杂度 $\mathcal{O}(n\log^2 n)$。
- 如果没有截断的情况，直接对点值求 k 次方也是正确的。
- 若 $A(x)$ 的常数项为 1，即求 $e^{k\ln A(x)}$，k 对模数 P 取模。
- 若 $A(x)$ 的常数项不为 1，找到第一个系数非 0 的项，设为 $a_0=[x^t]A(x)$，则所求变为 $a_0^kx^{tk}\left(\frac{A(x)}{a_0x^t}\right)^k$，乘以 a^k 时 k 对 $\varphi(P)=P-1$ 取模。

```cpp
const int mod = 998244353;
const int inv2 = 499122177;
const int inv3 = 332748118;
int rev[N4], tw[N4], inv[N4]; //polyInt 中的 inv 要预处理（线性求逆元）

inline void operator += (vector<int> &a, vector<int> b)
{
    int n = b.size();
    a.resize(Max((int)a.size(), n));
    for (int i = 0; i < n; ++i)
        add(a[i], b[i]);
}

inline void operator -= (vector<int> &a, vector<int> b)
{
    int n = b.size();
    a.resize(Max((int)a.size(), n));
    for (int i = 0; i < n; ++i)
        dec(a[i], b[i]);
}

inline void operator *= (vector<int> &a, int k)
{
    if (k == -1)
    {
        int n = a.size();
```

```
        for (int i = 0; i < n; ++i)
            if (a[i])
                a[i] = mod - a[i];
    }
    else
    {
        int n = a.size();
        for (int i = 0; i < n; ++i)
            a[i] = 1ll * k * a[i] % mod;
    }
}

inline void DFT(vector<int> &a, int opt)
{
    int n = a.size(), g = opt == 1 ? 3 : inv3;
    for (int i = 0; i < n; ++i)
        if (i < rev[i])
            std::swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k <<= 1)
    {
        int w = quick_pow(g, (mod - 1) / (k << 1));
        tw[0] = 1;
        for (int j = 1; j < k; ++j)
            tw[j] = 1ll * tw[j - 1] * w % mod;
        for (int i = 0; i < n; i += k << 1)
        {
            for (int j = 0; j < k; ++j)
            {
                int u = a[i + j],
                    v = 1ll * tw[j] * a[i + j + k] % mod;
                add(a[i + j] = u, v);
                dec(a[i + j + k] = u, v);
            }
        }
    }
    if (opt == -1)
    {
        int inv_n = quick_pow(n, mod - 2);
        for (int i = 0; i < n; ++i)
            a[i] = 1ll * a[i] * inv_n % mod;
    }
}

inline void polyMul(vector<int> &a, vector<int> b)
{
    if (!a.size() || !b.size())
    {
        a.clear();
        return ;
    }
    int m = 0, _n = a.size() + b.size() - 2, n;
    for (n = 1; n <= _n; n <<= 1)
        ++m;
    a.resize(n);
    b.resize(n);
    for (int i = 1; i < n; ++i)
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (m - 1));
    DFT(a, 1); DFT(b, 1);
```

```
    for (int i = 0; i < n; ++i)
        a[i] = 1ll * a[i] * b[i] % mod;
    DFT(a, -1);
    a.resize(_n + 1);
}

inline void solve(int l, int r)
{
    if (l == r)
        return ;
    int mid = l + r >> 1;
    solve(l, mid);
    vector<int> a(mid - l + 1), b(r - l + 1);
    for (int i = l; i <= mid; ++i)
        a[i - l] = f[i];
    for (int i = 0; i < r - l + 1; ++i)
        b[i] = g[i];
    polyMul(a, b);
    for (int i = mid + 1; i <= r; ++i)
        add(f[i], a[i - l]);
    solve(mid + 1, r);
}

inline void polyDer(vector<int> &a)
{
    int n = a.size();
    for (int i = 0; i < n; ++i)
        a[i] = 1ll * (i + 1) * a[i + 1] % mod;
    a.pop_back();
}

inline void polyInt(vector<int> &a)
{
    int n = a.size();
    a.push_back(0);
    for (int i = n - 1; i >= 0; --i)
        a[i + 1] = 1ll * inv[i + 1] * a[i] % mod;
    a[0] = 0;
}

//以下各函数 vector<int> &b 在传入之前需保证为空

inline void polyInv(vector<int> a, vector<int> &b)
{
    int n = a.size();
    b.push_back(quick_pow(a[0], mod - 2));
    vector<int> c;
    int m = 1, m2, k;
    while (m < n)
    {
        m <<= 1, k = 0;
        for (m2 = 1; m2 < (m << 1); m2 <<= 1)
            ++k;
        for (int i = 1; i < m2; ++i)
            rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (k - 1));
        b.resize(m2); c.resize(m2);
        for (int i = 0; i < m2; ++i)
            c[i] = i < n && i < m ? a[i] : 0;
```

```
        DFT(b, 1); DFT(c, 1);
        for (int i = 0; i < m2; ++i)
        {
            int tmp = b[i];
            add(b[i], tmp);
            dec(b[i], 1ll * c[i] * tmp % mod * tmp % mod);
        }
        DFT(b, -1);
        b.resize(m);
    }
    b.resize(n);
}

/*
polyInv's example:
5
1 6 3 4 9
1 998244347 33 998244169 1020

polyLn's example:
6
1 927384623 878326372 3882 273455637 998233543
0 927384623 817976920 427326948 149643566 610586717

polyExp's example:
reversed
*/

inline void polyLn(vector<int> a, vector<int> &b)
{
    int n = a.size();
    polyInv(a, b);
    polyDer(a);
    polyMul(b, a);
    b.resize(n - 1);
    polyInt(b);
}

inline void polyExp(vector<int> a, vector<int> &b)
{
    int n = a.size();
    b.push_back(1);
    vector<int> c;
    int m = 1;
    while (m < n)
    {
        m <<= 1;
        c.clear();
        b.resize(m);
        polyLn(b, c);
        for (int i = 0; i < m; ++i)
            add(c[i] = mod - c[i], i < n ? a[i] : 0);
        add(c[0], 1);
        polyMul(b, c);
        b.resize(m);
    }
    b.resize(n);
}
```

```cpp

// 调用该函数前应先特判因前若干项为 0 导致结果全为 0 的情况
// k1 是指数对 mod 取模后的值, k2 是指数对 mod - 1 取模后的值
// 指数较小时两者可合并

inline void polyPow(vector<int> &a, int k1, int k2)
{
    int n = a.size(), st = 0;
    while (st < n && !a[st])
        ++st;
    for (int i = st; i < n; ++i)
        a[i - st] = a[i];
    for (int i = n - st; i < n; ++i)
        a[i] = 0;
    a.resize(n);

    int a0 = a[0], inv_a0 = quick_pow(a0, mod - 2);
    for (int i = 0; i < n; ++i)
        a[i] = 1ll * a[i] * inv_a0 % mod;
    vector<int> b;
    polyLn(a, b);
    for (int i = 0; i < n; ++i)
        b[i] = 1ll * b[i] * k1 % mod;
    a.clear();
    polyExp(b, a);

    a.resize(n);
    a0 = quick_pow(a0, k2);
    for (int i = 0; i < n; ++i)
        a[i] = 1ll * a[i] * a0 % mod;
    for (int i = n - 1; i >= 0; --i)
        if (i + 1ll * st * k1 < n)
            a[i + 1ll * st * k1] = a[i];
    for (int i = 0, im = Min((ll)n, 1ll * st * k1); i < im; ++i)
        a[i] = 0;
}

inline void polySqrt(vector<int> a, vector<int> &b)
{
    int n = a.size();
    b.push_back(1);
    vector<int> c, d;
    int m = 1;
    while (m < n)
    {
        m <<= 1;
        b.resize(m);
        c.clear();
        polyInv(b, c);
        d.resize(m);
        for (int i = 0; i < m; ++i)
            d[i] = i < n ? a[i] : 0;
        polyMul(c, d);
        for (int i = 0; i < m; ++i)
            b[i] = 1ll * inv2 * (c[i] + b[i]) % mod;
    }
    b.resize(n);
}
```

生成函数

OGF

- 即一般生成函数 $A(x)=\sum_{i\ge 0} a_i x^i$。
- 常用公式：$\frac{1}{1-x}=\sum_{i\ge 0} x^i$, $\frac{1}{(1-x)^2}=\sum_{i\ge 0}(i+1)x^i$, $\frac{1}{(1-ax)^m}=\sum_{i\ge 0}\binom{i+m-1}{m-1}a^i x^i$
- 推导上述公式尽量用消项法或组合意义，求导/积分则较为麻烦。
- **结论** $\frac{1}{1-y}\left(\frac{y}{1-y}\right)^k=\sum_{n\ge 0}\binom{n}{k}y^n$

> **证明**
>
> $$\begin{aligned}A(x,y)&=\sum_{k\ge 0}\left(\sum_{n\ge 0}\binom{n}{k}y^n\right)x^k\\&=\sum_{n\ge 0}\sum_{k\ge 0}\binom{n}{k}x^k y^n\\&=\sum_{n\ge 0}[(1+x)y]^n\\&=\frac{1}{1-y-xy}\\&=\frac{1}{1-y}\frac{1}{1-\dfrac{y}{1-y}x}\\&=\sum_{k\ge 0}\frac{1}{1-y}\left(\frac{y}{1-y}\right)^k x^k\end{aligned}$$

典例 [Bobo String Count]

题目大意

- 给定一个 01 串 t，求在长度为 n 且 t 恰好出现 k 次的 01 串个数，答案对 998244353 取模。
- $0\le k\le n$，$1\le n,|t|\le 10^5$。

解法

- 令 $m=|t|$，设 g_i 表示长度为 $i+m$ 的字符串长度为 m 的前后缀均为 t 的方案数，则：
  - 若 $i<m$，若 $m-i$ 是 t 的 border，则 g_i 为 1，否则为 0。
  - 若 $i\ge m$，则 $g_i=2^{i-m}$。
- 设 h_i 表示长度为 $i+m$ 的字符串长度为 m 的前后缀均为 t 且中间不再有 t 出现的方案数，则根据容斥原理有：

$$h_i=g_i-\sum_{j=1}^{i-1}h_j g_{i-j}$$

- 设 $H(x)=\sum_{i\ge 1}h_i x^i$，$G(x)=\sum_{i\ge 0}g_i x^i$，则有：

$$G=HG+1\Leftrightarrow H=1-\frac{1}{G}$$

- 设 p_i 表示长度为 $i+m$ 的字符串前（后）缀为 t 的方案数，同样根据容斥原理有：

$$p_i=2^i-\sum_{j=1}^{i}h_j 2^{i-j}$$

- 设 $P(x)=\sum\limits_{i\geq 0}p_ix^i$，讨论 k:
  - 若 $k=0$，则答案为 $2^n-\sum\limits_{j=0}^{n-m}p_j2^{n-m-j}$，时间复杂度 $\mathcal{O}(n\log n)$。
  - 若 $k>0$，则答案为 $[x^{n-m}]P^2H^{k-1}$，时间复杂度 $\mathcal{O}(n\log n\log k)$。

EGF

- 即指数型生成函数 $A(x)=\sum\limits_{i\geq 0}\frac{a_i}{i!}x^i$。
- 若将 B 划分为若干个**非空无序集合** A，则 B 与 A 的 $\mathbb{EGF}$ 的关系为（$A(x)$ 常数项必须为 0，$B(x)$ 常数项必须为 1）：

$$B(x)=\sum_{i\geq 0}\frac{A(x)^i}{i!}=e^{A(x)},\ A(x)=\ln B(x)$$

  - 例如若 B 为排列，A 为置换，则

$$B(x)=\sum_{i\geq 0}\frac{i!}{i!}x^i=\sum_{i\geq 0}x^i=\frac{1}{1-x}$$
$$A(x)=\sum_{i\geq 1}\frac{(i-1)!}{i!}x^i=\sum_{i\geq 1}\frac{x^i}{i}=\ln\frac{1}{1-x}$$

  - 常见的例子即可通过 n 个点有标号无向图的 $\mathbb{EGF}$ 取 $\ln$ 得到 n 个点有标号无向连通图的 $\mathbb{EGF}$，对 n 个点有标号无向树的 $\mathbb{EGF}$ 取 $\exp$ 即可得到 n 个点有标号无向森林的 $\mathbb{EGF}$。
- 常用公式（即 Taylor 级数）：

$$\frac{e^x+e^{-x}}{2}=\sum_{i\geq 0\text{且}i\text{为偶数}}\frac{x^i}{i!},\ \frac{e^x-e^{-x}}{2}=\sum_{i\geq 0\text{且}i\text{为奇数}}\frac{x^i}{i!}$$
$$\ln(1+x)=\sum_{i\geq 1}\frac{(-1)^{i+1}}{i}x^i,\ \ln(1-x)=-\sum_{i\geq 1}\frac{x^i}{i}$$
$$(1+x)^\alpha=1+\sum_{i\geq 1}\frac{\prod\limits_{j=\alpha-i+1}^{\alpha}j}{i!}x^i$$

典例1 CF891E

题目大意

- 有 n 个数 $a_1,a_2,\dots,a_n$，要进行 k 次操作，每次随机选择一个数 $x\in[1,n]$，把 a_x 减 1，将答案增加除 a_x 外所有数的乘积。
- 求最终答案的期望。

解法

- 设 k 次操作后 a_i 变成了 a_i-b_i，实际每次操作的贡献是操作前后序列乘积的差，**即贡献序列是序列乘积的差分**，因而**总的贡献就是初始序列的乘积减去最终序列的乘积**，即求 $\prod\limits_{i=1}^{n}a_i-\prod\limits_{i=1}^{n}(a_i-b_i)$ 的期望。
- 显然只要考虑后半部分，用 $\mathbb{EGF}$ 推导所有方案的贡献之和，再除以总方案数即为期望。

$$F_i(x)=\sum_{j\geq 0}\frac{a_i-j}{j!}x^j=(a_i-x)e^x$$
$$F(x)=\prod_{i=1}^{n}F_i(x)=e^{nx}\prod_{i=1}^{n}(a_i-x)$$

- $F(x)$ 后半部分可以暴力卷积，同时我们只关心 $[x^k]F(x)$， 将前半部分展开后直接计算即可。

典例2 有标号（弱连通）DAG计数

题目大意

- 对 n 个点的有标号（弱连通）DAG 计数。
- $n \le 10^5$，答案对 998244353 取模。

解法

- 设 f_n 表示 n 个点有标号 DAG 的个数，考虑枚举入度为 0 的点的个数 j 进行容斥，则这 j 个点可与剩下 $i-j$ 个点任意连边，有：

$$f_i = \sum_{j=1}^{i} (-1)^{j+1} \binom{i}{j} 2^{j(i-j)} f_{i-j}$$

- 由 $j(i-j)$ 组合意义可知 **（常用的变换技巧）**：

$$j(i-j) = \binom{i}{2} - \binom{j}{2} - \binom{i-j}{2}$$

- 代入后得到：

$$f_i = \sum_{j=1}^{i} (-1)^{j+1} \frac{i!}{j!(i-j)!} \frac{2^{\binom{i}{2}}}{2^{\binom{j}{2}} 2^{\binom{i-j}{2}}} f_{i-j}$$

$$\frac{f_i}{i! 2^{\binom{i}{2}}} = \sum_{j=1}^{i} \frac{(-1)^{j+1}}{j! 2^{\binom{j}{2}}} \frac{f_{i-j}}{(i-j)! 2^{\binom{i-j}{2}}}$$

- 设 $F(x) = \sum\limits_{i \ge 0} \frac{f_i}{i! 2^{\binom{i}{2}}} x^i$，$G(x) = \sum\limits_{i \ge 1} \frac{(-1)^{i+1}}{i! 2^{\binom{i}{2}}} x^i$，上式写为：

$$F = FG + 1 \Leftrightarrow F = \frac{1}{1-G}$$

- 再设 h_n 表示 n 个点有标号弱连通 DAG 的个数，$P(x) = \sum\limits_{i \ge 0} \frac{f_i}{i!} x^i, Q(x) = \sum\limits_{i \ge 0} \frac{h_i}{i!} x^i$，由多项式 $\exp$ 的意义可知：

$$Q(x) = \ln P(x)$$

- 总时间复杂度 $\mathcal{O}(n \log n)$。

常见技巧

- 将生成函数 $A(x)$ 乘上 $\frac{1}{1-x}$ 相当于乘上 $\sum\limits_{i \ge 0} x^i$， 实际上就是对 $A(x)$ 的系数求前缀和，乘 $(\frac{1}{1-x})^k$ 即求 k 次前缀和，参见组合数学高阶前缀和部分。
- 欲求 $\prod\limits_{j=1}^{m} (1 - x^{c_j})$，其中 $\sum\limits_{j=1}^{m} c_i = n$，通常有两种处理方法：
  - 分治 **NTT**，时间复杂度 $\mathcal{O}(n \log n \log m)$。
  - 根据 Taylor 级数，预处理 cnt_i 表示数组 c 中 i 的出现次数，

$$\begin{aligned}\prod_{j=1}^{m}(1-x^{c_j}) &= \exp\sum_{j=1}^{m}\ln(1-x^{c_j})\\ &= \exp\left(-\sum_{j=1}^{m}\sum_{i\ge 1}\frac{x^{c_j i}}{i}\right)\\ &= \exp\left(-\sum_{i=1}^{n}\sum_{i|j}\frac{i\mathrm{cnt}_i x^j}{j}\right)\end{aligned}$$

时间复杂度 $\mathcal{O}(n\log n)$。

集合幂级数

- 类比数列的生成函数，设全集 $U=\{1,2,\dots,n\}$，定义集合幂级数：

$$f=\sum_{S\in 2^U} f_S x^S$$

- 加减法定义为系数相加减，乘法定义为：

$$h=f*g\Leftrightarrow \sum_{S\in 2^U} h_S x^S=\left(\sum_{S\in 2^U} f_S x^S\right)\left(\sum_{S\in 2^U} g_S x^S\right)=\sum_{S\in 2^U}\left(\sum_{L\in 2^U}\sum_{R\in 2^U}[L*R=S]f_L g_R\right)x^S$$

- 当 `*` 取不同的运算时会产生不同的效果。

OR 卷积

- 取 `*` 为或运算，则有 $h_S=\sum\limits_{L\in 2^U}\sum\limits_{R\in 2^U}[L\cup R=S]f_L g_R$。
- 定义 f 的 **莫比乌斯变换** 为集合幂级数 $\hat{f}$，其中 $\hat{f}_S=\sum\limits_{T\subseteq S} f_T$，则 $\hat{h_S}=\hat{f}_S\hat{g_S}$。
- 由容斥原理，定义 $\hat{f}$ 的 **莫比乌斯反演** 为 f，其中 $f_S=\sum\limits_{T\subseteq S}(-1)^{|S|-|T|}\hat{f}_T$。
- 可将莫比乌斯变换视为高维前缀和，将莫比乌斯反演视为其逆过程，卷积只需要在对应位置相乘即可，时间复杂度 $\mathcal{O}(n2^n)$。

AND 卷积

- 取 `*` 为与运算，同 OR 卷积的区别仅在于 0/1 位置互换，其余过程完全一致。

XOR 卷积

- 取 `*` 为异或运算，则有 $h_S=\sum\limits_{L\in 2^U}\sum\limits_{R\in 2^U}[L\oplus R=S]f_L g_R$。
- 注意到 $\frac{1}{2^n}\sum\limits_{T\in 2^U}(-1)^{|T\cap S|}=[S=\varnothing]$，证明即考虑 S 不为空时，任取 $v\in S$，T 和 $T\oplus v$ 产生贡献的和恒为 0。
- 因而可化简得到：

$$\begin{aligned}h_S &= \sum_{L\in 2^U}\sum_{R\in 2^U}[L\oplus R\oplus S=\varnothing]f_L g_R\\ &= \sum_{L\in 2^U}\sum_{R\in 2^U}\frac{1}{2^n}\sum_{T\in 2^U}(-1)^{|T\cap(L\oplus R\oplus S)|}f_L g_R\\ &= \sum_{L\in 2^U}\sum_{R\in 2^U}\frac{1}{2^n}\sum_{T\in 2^U}(-1)^{|T\cap L|}(-1)^{|T\cap R|}(-1)^{|T\cap S|}f_L g_R\\ &= \frac{1}{2^n}\sum_{T\in 2^U}(-1)^{|T\cap S|}\left(\sum_{L\in 2^U}(-1)^{|T\cap L|}f_L\right)\left(\sum_{R\in 2^U}(-1)^{|T\cap R|}g_R\right)\end{aligned}$$

- 根据上述结果（可取 $g = x^{\varnothing}$）定义 f 的 **沃尔什变换** 为集合幂级数 $\hat{f}$，$\hat{f}$ 的 **沃尔什逆变换** 为 f，其中：

$$\hat{f}_S = \sum_{T\in 2^U} (-1)^{|S\cap T|} f_T \Leftrightarrow f_S = \frac{1}{2^n}\sum_{T\in 2^U}(-1)^{|S\cap T|}\hat{f}_T$$

- 此时便满足 $\hat{h}_s = \hat{f}_s\hat{g}_s$。
- 考虑怎样快速求一个集合幂级数 f 的沃尔什变换，沃尔什逆变换只需乘上 $\frac{1}{2^n}$ 的系数即可。
- 设 $\hat{f_S}^{(i)} = \sum\limits_{S\oplus T\subseteq\{1,2,\dots,i\}} (-1)^{|S\cap T\cap\{1,2,\dots,i\}|}$，则有 $\hat{f_S}^{(0)} = f_S, \hat{f_S}^{(n)} = \hat{f}_S$，不难得到：

$$\hat{f_S}^{(i)} = \hat{f_S}^{(i-1)} + \hat{f}^{(i-1)}_{S\cup\{i\}} \quad \hat{f}^{(i)}_{S\cup\{i\}} = \hat{f_S}^{(i-1)} - \hat{f}^{(i-1)}_{S\cup\{i\}}$$

- 顺序枚举 i 从 1 到 n 迭代即可，时间复杂度 $\mathcal{O}(n2^n)$。

子集卷积

- 定义 $h = f \cdot g$ 表示 f 和 g 的子集卷积，其中 h 为集合幂级数：

$$h_S = \sum_{L\in 2^U}\sum_{R\in 2^U}[L\cup R = S][L\cap R = \varnothing] f_L g_R$$

- 注意到 $[L\cup R = S][L\cap R = \varnothing] = [L\cup R = S][|L| + |R| = |S|]$，可设集合占位幂级数（其中 $z^a * z^b = z^{a+b}$）：

$$\sigma = \sum_{S\in 2^U} f_S z^{|S|} x^S$$

- 对集合幂级数做 OR 卷积时用快速莫比乌斯变换优化，对 z 做卷积时暴力即可，时间复杂度 $\mathcal{O}(n^2 2^n)$。
- 也可将上述四种卷积变换的结果理解成点值，做多项式运算时只需在最外层变换和反演（逆变换）一次。

```
const int mod = 1e9 + 9;
const int inv2 = mod + 1 >> 1;

inline void orFMT(int *f, int n, int opt)
{
    int s = 1 << n;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < s; ++j)
            if (!(j >> i & 1))
                opt == 1 ? add(f[1 << i | j], f[j]) : dec(f[1 << i | j], f[j]);
}

inline void andFMT(int *f, int n, int opt)
{
    int s = 1 << n;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < s; ++j)
            if (!(j >> i & 1))
                opt == 1 ? add(f[j], f[1 << i | j]) : dec(f[j], f[1 << i | j]);
}

inline void FWT(int *f, int n, int opt)
```

```
{
    int s = 1 << n;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < s; ++j)
            if (!(j >> i & 1))
            {
                int tj = 1 << i | j;
                int l = f[j],
                    r = f[tj];
                f[j] = f[tj] = l;
                add(f[j], r);
                dec(f[tj], r);
            }
    if (opt == -1)
    {
        int inv_s = 1;
        for (int i = 0; i < n; ++i)
            inv_s = 1ll * inv2 * inv_s % mod;
        for (int i = 0; i < s; ++i)
            f[i] = 1ll * f[i] * inv_s % mod;
    }
}

// 以下函数传入后 f,g 内的信息都将改变，结果存入 f

inline void polyBit(void (*trans)(int*, int, int), int *f, int *g, int n)
{   // And Or Xor 卷积
    trans(f, n, 1);
    trans(g, n, 1);
    int s = 1 << n;
    for (int i = 0; i < s; ++i)
        f[i] = 1ll * f[i] * g[i] % mod;
    trans(f, n, -1);
}

inline void polySubset(int *f, int *g, int n)
{
    static int tf[N][S], tg[N][S], th[N][S], cnt[S];
    int s = 1 << n;
    for (int i = 1; i < s; ++i)
        cnt[i] = cnt[i ^ (i & -i)] + 1;
    for (int i = 0; i < s; ++i)
        tf[cnt[i]][i] = f[i], tg[cnt[i]][i] = g[i];
    for (int i = 0; i <= n; ++i)
        orFMT(tf[i], n, 1), orFMT(tg[i], n, 1);
    for (int i = 0; i <= n; ++i)
        for (int j = 0; j <= i; ++j)
            for (int k = 0; k < s; ++k)
                th[i][k] = (1ll * tf[j][k] * tg[i - j][k] + th[i][k]) % mod;
    for (int i = 0; i <= n; ++i)
        orFMT(th[i], n, -1);
    for (int i = 0; i < s; ++i)
        f[i] = th[cnt[i]][i];

    for (int i = 0; i < s; ++i)
        cnt[i] = 0;
    for (int i = 0; i <= n; ++i)
        for (int j = 0; j < s; ++j)
```

```
            tf[i][j] = tg[i][j] = th[i][j] = 0;
}
```