容斥与反演

单位根反演

将$[i|n]$展开为ω_n：

设$i \mod n = r$，则当$r \neq 0$时有

$$\sum_{j=0}^{n-1} \omega_n^{ij} = \frac{\omega_n^{in} - 1}{\omega_n^i - 1} = \frac{(\omega_n^n)^i - 1}{\omega_n^i - 1} = 0$$

否则有

$$\sum_{j=0}^{n-1} \omega_n^{knj} = n$$

因此有

$$[i|n] = \frac{1}{n} \sum_{j=0}^{n-1} \omega_n^{ij}$$

莫比乌斯反演

将$[n = 1]$展开为μ：

$$[n = 1] = \sum_{d|n} \mu(d) = \sum_{d} [d|n] \mu(d)$$

一般偏序集上的容斥与反演

在一般偏序集上有一些二元函数$f : S \times S \to R$，其中$<$是S上的偏序，R是任意一个环。这些函数满足性质

$$f(x, y) \neq 0 \to x \leq y$$

可以定义这些函数之间的卷积：

$$h(x, y) = \sum_{x \leq z \leq y} f(x, z) g(z, y)$$

注：不满足$x \leq y$时不存在满足$x \leq z \leq y$的z，此时$h(x, y) = 0$。

可以验证这种卷积满足结合律，即

$$(f * g) * h = f * (g * h)$$

定义(δ函数)：

$$\delta(x, y) = [x = y]$$

不难验证δ为卷积的单位元。

对于满足$f(x, x) \neq 0$的函数可以定义其左逆g满足$g * f = \delta$

$$\sum_{x \leq z \leq y} g(x, z) f(z, y) = \delta(x, y)$$

因此有

$$g(x,y)=\begin{cases}1/f(y,y) & x=y\\ -\dfrac{1}{f(y,y)}\displaystyle\sum_{x\le z<y}g(x,z)f(z,y) & \end{cases}$$

可以用类似的方式定义右逆h满足$f*h=\delta$。由此有：

$$g=g*\delta=g*(f*h)=(g*f)*h=\delta*h=h$$

因此若函数f有逆则其逆唯一且同时为左逆和右逆。

定义(ζ函数)：

$$\zeta(x,y)=[x\le y]$$

定义(μ函数)：

$$\sum_{x\le z\le y}\mu(x,z)\zeta(z,y)=\delta(x,y)=[x=y]$$

μ为ζ的逆。

当$(S,<)$为$(\mathbb{N}^+,|)$时，即集合为正整数而偏序关系为整除关系时，对每个$f:S\times S\to R$定义对应的一元函数

$$f(x,y)=f(y/x)$$

则有卷积

$$(f*g)(x,y)=\sum_{x\le z\le y}f(x,z)g(z,y)=\sum_{x|z|y}f(z/x)g(y/z)=\sum_{d|(x/y)}f(d)g((x/y)/d)$$

令$n=x/y$，则

$$(f*g)(n)=\sum_{d|n}f(d)g(n/d)$$

此时卷积即为狄利克雷卷积，ζ函数即为常函数1，μ函数即为数论中的μ函数。

当$(S,<)$为$(2^X,\subset)$时，即集合为某个集合X的所有子集而偏序关系为包含关系时，对每个$f:2^X\times 2^X\to R$定义对应的一元函数

$$f(S,T)=f(T-S)$$

则有卷积

$$(f*g)(S,T)=\sum_{S\subseteq U\subseteq T}f(S,U)g(U,T)=\sum_{S\subseteq U\subseteq T}f(U-S)g(T-U)=\sum_{U\subseteq S-T}f(U)g((T-S)-U)=\sum_{T\subseteq S}f(T)g(S-T)$$

令$U=T-S$，则

$$(f*g)(U)=\sum_{S\subseteq U}f(S)g(U-S)$$

此时卷积即为子集卷积，ζ即为常函数1，μ函数为$\mu(S)=(-1)^{|S|}$。

二项式反演

此即$(2^X,\subset)$上的一种特殊形式。若

$$f(S)=a_{|S|},g(S)=b_{|S|}=\sum_{T\subseteq S}f(T)$$

则有

$$b_i=\sum_{j=0}^{i}\binom{i}{j}a_j,a_i=\sum_{j=0}^{i}\binom{i}{j}(-1)^{i-j}b_j$$

斯特林反演

生成函数

形式幂级数

$$\frac{1}{1-x}=\sum_{i=0}^{\infty}x^i=1+x+x^2+\cdots$$

$$\frac{x}{(1-x)^2}=\sum_{i=0}^{\infty}nx^n=x+2x^2+3x^3+\cdots$$

$$\frac{x(x+1)}{(1-x)^3}=\sum_{i=0}^{\infty}n^2x^n=x+4x^2+9x^3+\cdots$$

$$\exp x=\sum_{i=0}^{\infty}\frac{x^i}{i!}=1+x+\frac{x^2}{2}+\frac{x^3}{6}+\cdots$$

$$\log(1+x)=\sum_{i=1}^{\infty}\frac{(-1)^{i+1}}{i}x^i=x-\frac{x^2}{2}+\frac{x^3}{3}+\cdots$$

$$(1+x)^n=\sum_{i=0}^{n}\binom{n}{i}x^i$$

多元幂级数

$$\frac{1}{1-(1+x)y}=\sum_{i=0}^{\infty}(1+x)^iy^i=\sum_{i=0}^{\infty}\sum_{j=0}^{i}\binom{i}{j}x^jy^i$$

常见变换

	$xf(x)$	$(f(x)-a_0)/x$	$f'(x)$	$\int f(x)dx$
$[x^n]$	a_{n-1}	a_{n+1}	$(n+1)a_{n+1}$	a_{n-1}/n
$\left[\frac{x^n}{n!}\right]$	na_{n-1}	$a_{n+1}/(n+1)$	a_{n+1}	a_{n-1}

OGF第i项乘i：

$$(Tf)(x)=xf'(x)=\sum_{i=0}^{\infty}ia_ix^i$$

设$(T^{(n)}f)(x)=\sum\limits_{k=0}^{n}c_{n,k}x^kf^{(k)}(x)$，则

$$\begin{aligned}(T^{(n+1)}f)(x)&=x(T^{(n)}f)'(x)=x\left[\sum_{k=0}^{n}c_{n,k}(x^kf^{(k+1)}(x)+kx^{k-1}f^{(k)}(x)\right]\\&=\sum_{k=0}^{n}c_{n,k}(x^{k+1}f^{(k+1)}(x)+kx^kf^{(k)}(x))\\&=\sum_{k=0}^{n}c_{n,k}kx^kf^{(k)}(x)+\sum_{k=1}^{n+1}c_{n,k-1}x^kf^{(k)}(x)\end{aligned}$$

考虑到$c_{n,n+1}=0$，因此有

$$=\sum_{k=0}^{n+1}(kc_{n,k}+c_{n,k-1})x^kf^{(k)}(x)=\sum_{k=0}^{n+1}c_{n+1,k}x^kf^{(k)}(x)$$

即$c_{n,k}=kc_{n-1,k}+c_{n-1,k-1}$。结合初始条件有$c_{n,k}=\begin{Bmatrix}n\\k\end{Bmatrix}$。即

$$(T^{(n)}f)(x)=\sum_{k=0}^{n}\begin{Bmatrix}n\\k\end{Bmatrix}x^kf^{(k)}(x)$$

例：令$f(x)=(1-x)^{-1}$，即得

$$f^{(k)}(x)=\sum_{i=k}^{\infty}i^{\underline{k}}x^{i-k}$$

$$\begin{aligned}(T^{(k)}f)(x)&=\sum_{i=0}^{\infty}i^kx^i=\sum_{i=0}^{k}\begin{Bmatrix}k\\i\end{Bmatrix}x^i\sum_{j=i}^{\infty}j^{\underline{i}}x^{j-i}=\sum_{i=0}^{k}\begin{Bmatrix}k\\i\end{Bmatrix}\sum_{j=i}^{\infty}j^{\underline{i}}x^j\\&=\sum_{j=0}^{\infty}x^j\sum_{i=0}^{k}\begin{Bmatrix}k\\i\end{Bmatrix}j^{\underline{i}}=\sum_{j=0}^{\infty}x^j\sum_{i=0}^{k}\begin{Bmatrix}k\\i\end{Bmatrix}\binom{j}{i}i!\end{aligned}$$

因此有

$$j^k=\sum_{i=0}^{k}\begin{Bmatrix}k\\i\end{Bmatrix}\binom{j}{i}i!$$

卷上一个$(1+x)^{-1}$即得

$$\sum_{j=0}^{n}j^k=\sum_{j=0}^{n}\sum_{i=0}^{k}\begin{Bmatrix}k\\i\end{Bmatrix}\binom{j}{i}i!=\sum_{i=0}^{k}\begin{Bmatrix}k\\i\end{Bmatrix}i!\sum_{j=0}^{n}\binom{j}{i}=\sum_{i=0}^{k}\begin{Bmatrix}k\\i\end{Bmatrix}i!\binom{n+1}{i+1}$$

一般生成函数(OGF)

$$f(x)=\sum_{i=0}^{\infty}a_ix^i$$

一般生成函数一般用于组合的计数（位置不可区分，或无标号）

OGF卷积的组合意义

考虑f和g的卷积：

$$h(x)=f(x)g(x)=\sum_{i=0}^{\infty}\left(\sum_{j=0}^{i}a_jb_{i-j}\right)x^i=\sum_{i=0}^{\infty}c_ix^i$$

可以理解为：有两类物品组合。第一类中大小为i的有a_i种，第二类中大小为i的有b_i种。则分别从第一类和第二类中分别选出两个组合将其合并，大小为i的有c_i种。

欧拉变换

大小为i的组合有a_i种，现在计算总大小为i的由一个或多个组合组成的组合方案数量。

因为组合之间不可区分，所以可以枚举大小为i的组合中每一个用了多少次。即

$$\mathscr{E}(f)(x)=\left(1+x+x^2+\cdots\right)^{a_1}\left(1+x^2+x^4+\cdots\right)^{a_2}\cdots=\prod_{i=1}^{\infty}\left(\frac{1}{1-x^i}\right)^{a_i}$$

$$=\exp\left(\sum_{i=0}^{\infty}a_i\ln\frac{1}{1-x^i}\right)=\exp\left(\sum_{i=0}^{\infty}-a_i\ln\left(1-x^i\right)\right)$$

$$=\exp\left(\sum_{i=0}^{\infty}a_i\sum_{j=1}^{\infty}\frac{x^{ij}}{j}\right)=\exp\left(\sum_{j=1}^{\infty}\frac{1}{j}\sum_{i=0}^{\infty}a_ix^{ij}\right)=\exp\left(\sum_{j=1}^{\infty}\frac{f(x^j)}{j}\right)$$

指数生成函数(EGF)

$$f(x)=\sum_{i=0}^{\infty}a_i\frac{x^i}{i!}$$

指数生成函数一般用于排列的计数（位置可区分，或有标号）

EGF卷积的组合意义

考虑f和g的卷积：

$$h(x)=f(x)g(x)=\sum_{i=0}^{\infty}\left(\sum_{j=0}^{i}\frac{a_j}{j!}\frac{b_{i-j}}{(i-j)!}\right)x^i=\sum_{i=0}^{\infty}\left(\sum_{j=0}^{i}\binom{i}{j}a_jb_{i-j}\right)\frac{x^i}{i!}=\sum_{i=0}^{\infty}c_ix^i$$

可以理解为：有两类物品排列。第一类中大小为i的有a_i种，第二类中大小为i的有b_i种。则分别从第一类和第二类中分别选出两个排列将其合并，大小为i的有c_i种。

注：先钦定来自第一类的物品放在哪些位置，来自第二类的物品放在剩下那些位置，因此有个二项式系数。

$\exp f(x)$的组合意义

大小为i的排列有a_i种，现在计算大小为i的由数个排列组成的排列方案。

由k个排列组成的方案数的生成函数是

$$\frac{f(x)^k}{k!}$$

除以$k!$的原因是排列之间没有先后顺序。

枚举k可以得到

$$g(x)=\sum_{k=1}^{\infty}\frac{f(x)^k}{k!}=\exp f(x)$$

狄利克雷生成函数(DSGF)

$$f(s)=\sum_{i=1}^{\infty}\frac{a_i}{i^s}$$

例：常数序列1的DSGF是黎曼ζ函数。

DSGF卷积的组合意义

$$h(s)=f(s)g(s)=\sum_{i=1}^{\infty}\sum_{j|i}\frac{a_jb_{i/j}}{j^s(i/j)^s}=\sum_{i=1}^{\infty}\frac{1}{i^s}\sum_{j|i}a_jb_{i/j}$$

不难发现这个就是狄利克雷卷积。

若$\lambda(i)=a_i$是积性的，则

$$f(s)=\sum_{i=1}^{\infty}\frac{a_i}{i^s}=\prod_{p\in\mathbb{P}}(\lambda(1)+\lambda(p)+\lambda(p^2)+\cdots)=\prod_{p\in\mathbb{P}}\sum_{i=0}^{\infty}a_{p^i}$$

卷积相关

基本形式	变形
$c_i=\sum_{j=0}^{i}a_jb_{i-j}$	$c_i=\sum_{j=i}^{n}a_jb_{j-i}\overset{i=n-i'}{\Longrightarrow}c_{n-i}=\sum_{j=0}^{i}a_{n-j}b_{i-j}$

多项式基础

快速卷积原理

给定$f(x),g(x)$，计算$h(x)=f(x)g(x)$。

因为一个$n-1$次多项式的系数可以被其在n个不同位置上的取值唯一确定（考虑对其在n个位置上的取值列一个n元线性方程组，其系数矩阵即为范德蒙德矩阵，因范德蒙德行列式不为0当且仅当x_i两两不同，所以该方程组有唯一解），所以如果对于一个$n-1$次多项式我们能够快速求出$f(x)$在n个不同位置上的取值并根据其反推原多项式的系数，卷积即可快速完成。

快速傅里叶变换

快速傅里叶变换可在$O(n\log n)$内从$n-1$次多项式的n个系数推出多项式在n个不同位置（必须是单位根的幂）的取值。

设ω_n满足$\omega_n^n=1$且$\forall i\in\{1,2,\dots,n-1\}$，$\omega_n^i\neq 1$，我们选取的位置即为$x=\omega_n^i,i\in\{0,1,\dots,n-1\}$。

考虑利用单位根ω_n分治对系数序列进行变换，即求出$f(\omega_n^k)$，$k\in\{0,1,\dots,n-1\}$，这里设$n=2^w$。

这里若选取的域为$\mathbb{C}$，则$\omega_n=\exp 2\pi/n=\cos(2\pi/n)+i\sin(2\pi/n)$。若选取的域为$\mathbb{F}_p$，则$\omega_n=g^{P-1}/n$。（见NTT部分）

将序列按下标的奇偶分成两份，即

$$f_0(x)=\sum_{i=0}^{\frac{n}{2}-1}a_{2i}x^i,f_1(x)=\sum_{i=0}^{\frac{n}{2}-1}a_{2i+1}x^i$$

对每份分别进行FFT可得$f_0(\omega_{\frac{n}{2}}^k),f_1(\omega_{\frac{n}{2}}^k),k\in\{0,1,\dots,\frac{n}{2}-1\}$。

注：对f_0和f_1进行FFT时所用的单位根为$\omega_{\frac{n}{2}}=\omega_n^2$，即分治下去所得结果为$f_0,f_1$在$\omega_n^{2k},k\in\{0,1,\dots,\frac{n}{2}-1\}$上的取值。

$$f(\omega_n^k)=\sum_{i=0}^{n-1}a_i\omega_n^{ki}=\sum_{i=0}^{\frac{n}{2}-1}a_{2i}\omega_n^{2ki}+\sum_{i=0}^{\frac{n}{2}-1}a_{2i+1}\omega_n^{2ki+k}$$

$$=\sum_{i=0}^{\frac{n}{2}-1}a_{2i}\omega_n^{2ki}+\omega_n^k\sum_{i=0}^{\frac{n}{2}-1}a_{2i+1}\omega_n^{2ki}=f_0(\omega_n^{2k})+\omega_n^kf_1(\omega_n^{2k})$$

$$f(\omega_n^{k+\frac{n}{2}})=f(-\omega_n^k)=\sum_{i=0}^{n-1}a_i(-\omega_n^k)^i=\sum_{i=0}^{\frac{n}{2}-1}a_{2i}(-\omega_n^k)^{2i}+\sum_{i=0}^{\frac{n}{2}-1}a_{2i+1}(-\omega_n^k)^{2i+1}$$

$$=\sum_{i=0}^{\frac{n}{2}-1}a_{2i}\omega_n^{2ki}-\omega_n^k\sum_{i=0}^{\frac{n}{2}-1}a_{2i+1}\omega_n^{2ki}=f_0(\omega_n^{2k})-\omega_n^kf_1(\omega_n^{2k})$$

注：上述过程的$k\in\{0,1,\dots,\frac{n}{2}-1\}$

于是每层分治可以在$O(n)$复杂度内合并，总复杂度$O(n\log n)$。

```
typedef long long ll;

namespace FFT {

typedef long double dbl;
typedef complex<dbl> cplx;
const cplx I(0, 1);
const dbl pi = acos(-1);

const int W = 21, S = 1 << W;

int r[S], m, s; cplx w[S];

void init(int n) {
    for (s = 0, m = 1; m < n; m <<= 1) ++s;
    for (int i = 0; i != m; ++i) r[i] = (r[i >> 1] >> 1) | ((i & 1) << (s - 1));
    for (int i = 0; i <= m; ++i) w[i] = { cos(2 * pi * i / m), sin(2 * pi * i / m) };
}

void fft(cplx* p) {
    for (int i = 0; i != m; ++i) if (i < r[i]) swap(p[i], p[r[i]]);
    for (int i = 1, z = 0; i != m; i <<= 1, z++)
        for (int j = 0; j != m; j += (i << 1))
            for (int k = 0; k != i; k++) {
                cplx u = p[j + k], v = w[k<<s-z-1] * p[i + j + k];
                p[j + k] = u + v; p[i + j + k] = u - v;
            }

}

cplx pa[S], pb[S], pc[S], pd[S];
vector<int> conv(const vector<int>& a, const vector<int>& b) {
    int n1 = a.size(), n2 = b.size(), n3 = n1 + n2 - 1;
    init(n3);
```

```
    for (int i = 0; i < m; ++i) pa[i] = cplx(i < n1 ? a[i] : 0, i < n2 ? b[i] : 0);
    fft(pa);
    for (int i = 0; i < m; ++i) {
        int j = i ? m - i : 0;
        pb[i] = (conj(pa[j]) + pa[i]) * (conj(pa[j]) - pa[i]) * I / cplx(4);
    }
    fft(pb);
    vector<int> c(n3);
    for (int i = 0; i < n3; ++i) c[i] = round(pb[i ? m - i : 0].real() / m);
    return c;
}

vector<ll> conv2(const vector<int>& a, const vector<int>& b) {
    const int s = 15, msk = (1 << s) - 1;
    int n1 = a.size(), n2 = b.size(), n3 = n1 + n2 - 1; init(n3);
    for (int i = 0; i < m; ++i) {
        pa[i] = (i < n1 ? cplx(a[i] & msk, a[i] >> s) : cplx(0));
        pb[i] = (i < n2 ? cplx(b[i] & msk, b[i] >> s) : cplx(0));
    }
    fft(pa); fft(pb);
    for (int i = 0; i < m; ++i) {
        int j = i ? m - i : 0;
        pc[i] = (conj(pa[j]) + pa[i]) * pb[i] / cplx(2);
        pd[i] = (conj(pa[j]) - pa[i]) * pb[i] * I / cplx(2);
    }
    fft(pc); fft(pd);
    reverse(pc + 1, pc + m);
    reverse(pd + 1, pd + m);
    vector<ll> c(n3);
    for (int i = 0; i < n3; ++i) {
        pc[i] /= m; pd[i] /= m;
        ll ac = round(pc[i].real()), bc = round(pc[i].imag());
        ll ad = round(pd[i].real()), bd = round(pd[i].imag());
        c[i] = ac + ((bc + ad) << s) + (bd << (2 * s));
    }
    return c;
}

}  // namespace FFT
```

快速傅里叶逆变换

快速傅里叶逆变换可在$O(n\log n)$内从多项式在n个不同位置（必须是单位根的幂）的取值推出多项式的n个系数。

考虑对f利用单位根ω_n进行FFT得到g，再对g利用单位根ω_n^{-1}进行FFT得到h，并设g,h的系数分别为$\{b_j\},\{c_k\}$

$$b_j=\sum_{i=0}^{n-1}a_i\omega_n^{ij}$$

$$c_k=\sum_{j=0}^{n-1}b_j\omega_n^{-j}=\sum_{j=0}^{n-1}\left(\sum_{i=0}^{n-1}a_i\omega_n^{ij}\right)\omega_n^{-jk}=\sum_{i=0}^{n-1}a_i\sum_{j=0}^{n-1}\omega_n^{ij}\omega_n^{-jk}=\sum_{i=0}^{n-1}a_i\sum_{j=0}^{n-1}\omega_n^{(i-k)j}$$

注意到当$i=k$时有

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

当$i\neq k$时有

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

所以$c_k=na_k$。

注意到

$$b_j=\sum_{i=0}^{n-1}a_i\omega_n^{ij}=a_0+\sum_{i=1}^{n-1}a_{n-i}\omega_n^{(n-i)j}$$

$$=a_0+\sum_{i=1}^{n-1}a_{n-i}\omega_n^{nj-ij}=a_0+\sum_{i=1}^{n-1}a_{n-i}\omega_n^{-ij}=a_0+\sum_{i=1}^{n-1}a_{n-i}(\omega_n^{-1})^{ij}$$

因此逆变换相当于变换之后`reverse`除第一个元素之外的整个序列再除以n。

实数快速傅里叶变换

FFT可以用来快速计算卷积，一般的形式如下：

```
fft(a, 1); fft(b, 1);
for (int i = 0; i < n; ++i) c[i] = a[i] * b[i];
fft(c, -1);
```

这样需要进行3次FFT。

令$u_j=a_j+ib_j$，$v_j=a_j-ib_j$，于是

$$\mathscr{F}(u)_j=\sum_{k=0}^{n-1}(a_k+ib_k)(\cos(2\pi jk/n)+i\sin(2\pi jk/n))$$

$$=\sum_{k=0}^{n-1}(a_k\cos(2\pi jk/n)-b_k\sin(2\pi jk/n))+i(a_k\sin(2\pi jk/n)+b_k\cos(2\pi jk/n))$$

$$\mathscr{F}^{-1}(v)_j=\sum_{k=0}^{n-1}(a_k-ib_k)(\cos(2\pi jk/n)-i\sin(2\pi jk/n))$$

$$=\sum_{k=0}^{n-1}(a_k\cos(2\pi jk/n)-b_k\sin(2\pi jk/n))-i(a_k\sin(2\pi jk/n)+b_k\cos(2\pi jk/n))$$

因此有

$$\mathscr{F}(u)_j=\overline{\mathscr{F}^{-1}(v)_j}$$

同时由单位根的性质有

$$\mathscr{F}(u)_j=\sum_{k=0}^{n-1}u_k\omega_n^{jk}=\sum_{k=0}^{n-1}u_k(\omega_n^{-1})^{-jk}=\sum_{k=0}^{n-1}u_k(\omega_n^{-1})^{(n-j)k}=\mathscr{F}^{-1}(u)_{n-j}$$

因此可以通过一次FFT求出$\mathscr{F}(u)$与$\mathscr{F}(v)$。

注意到

$$\mathscr{F}(u)_j + \mathscr{F}(v)_j = 2\sum_{k=0}^{n-1} a_k(\cos(2\pi jk/n) + i\sin(2\pi jk/n))$$
$$\mathscr{F}(u)_j - \mathscr{F}(v)_j = 2\sum_{k=0}^{n-1} ib_k(\cos(2\pi jk/n) + i\sin(2\pi jk/n))$$

因此

$$\mathscr{F}(a)_j = \frac{\mathscr{F}(u)_j + \mathscr{F}(v)_j}{2}$$
$$\mathscr{F}(b)_j = -i\frac{\mathscr{F}(u)_j - \mathscr{F}(v)_j}{2}$$

因此可以通过一次FFT求出两个多项式的DFT结果。

当答案系数绝对值大到`long double`都遭不住的程度的时候可以将系数拆成高位和低位。如

$$u_i = a_i + eb_i, v_i = c_i + ed_i$$

其中有$e = 2^{16}$。则

$$(u * v)_i = \sum_{j=0}^{i}(a_j + eb_j)(c_{i-j} + ed_{i-j}) = \sum_{j=0}^{i}\left[a_jc_{i-j} + eb_jc_{i-j} + ea_jd_{i-j} + e^2b_jd_{i-j}\right]$$
$$= (a * c)_i + e(b * c)_i + e(a * d)_i + e^2(b * d)_i$$

可用四次FFT解决。

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

namespace FFT {

typedef long double dbl;
typedef complex<dbl> cplx;
const cplx I(0, 1);
const dbl pi = acos(-1);

const int W = 21, S = 1 << W;

int r[S], m, s; cplx w[S];

void init(int n) {
    for (s = 0, m = 1; m < n; m <<= 1) ++s;
    for (int i = 0; i != m; ++i) r[i] = (r[i >> 1] >> 1) | ((i & 1) << (s -
1));
    for (int i = 0; i <= m; ++i) w[i] = { cos(2 * pi * i / m), sin(2 * pi *
i / m) };
}

void fft(cplx* p) {
    for (int i = 0; i != m; ++i) if (i < r[i]) swap(p[i], p[r[i]]);
    for (int i = 1, z = 0; i != m; i <<= 1, z++)
        for (int j = 0; j != m; j += (i << 1))
            for (int k = 0; k != i; k++) {
                cplx u = p[j + k], v = w[k<<s-z-1] * p[i + j + k];
```

```
                p[j + k] = u + v; p[i + j + k] = u - v;
            }

}

cplx pa[S], pb[S], pc[S], pd[S];
vector<int> conv(const vector<int>& a, const vector<int>& b) {
    int n1 = a.size(), n2 = b.size(), n3 = n1 + n2 - 1;
    init(n3);
    for (int i = 0; i < m; ++i) pa[i] = cplx(i < n1 ? a[i] : 0, i < n2 ?
b[i] : 0);
    fft(pa);
    for (int i = 0; i < m; ++i) {
        int j = i ? m - i : 0;
        pb[i] = (conj(pa[j]) + pa[i]) * (conj(pa[j]) - pa[i]) * I / cplx(4);
    }
    fft(pb);
    vector<int> c(n3);
    for (int i = 0; i < n3; ++i) c[i] = round(pb[i ? m - i : 0].real() / m);
    return c;
}

vector<ll> conv2(const vector<int>& a, const vector<int>& b) {
    int n1 = a.size(), n2 = b.size(), n3 = n1 + n2 - 1;
    const int s = 15, msk = (1 << s) - 1; init(n3);
    for (int i = 0; i < m; ++i) {
        pa[i] = (i < n1 ? cplx(a[i] & msk, a[i] >> s) : cplx(0));
        pb[i] = (i < n2 ? cplx(b[i] & msk, b[i] >> s) : cplx(0));
    }
    fft(pa); fft(pb);
    for (int i = 0; i < m; ++i) {
        int j = i ? m - i : 0;
        pc[i] = (conj(pa[j]) + pa[i]) * pb[i] / cplx(2);
        pd[i] = (conj(pa[j]) - pa[i]) * pb[i] * I / cplx(2);
    }
    fft(pc); fft(pd);
    reverse(pc + 1, pc + m);
    reverse(pd + 1, pd + m);
    vector<ll> c(n3);
    for (int i = 0; i < n3; ++i) {
        pc[i] /= m; pd[i] /= m;
        ll ac = round(pc[i].real()), bc = round(pc[i].imag());
        ll ad = round(pd[i].real()), bd = round(pd[i].imag());
        c[i] = ac + ((bc + ad) << s) + (bd << (2 * s));
    }
    return c;
}

}  // namespace FFT
```

快速数论变换

在一般的FFT中，取$\omega_n = \cos\frac{2\pi}{n} + i\sin\frac{2\pi}{n}$并在复数域上进行运算即可。在系数较大时会产生精度问题，但使用`long double`可大幅度提升精度与TLE风险。

有时我们面对的问题是模素数意义下的，这时我们可以考虑直接在模素数有限域下解决快速傅里叶变换的问题。对于素数$p=2^kr+1$，模p域乘法群为$p-1$次循环群Z_{p-1}，且有子群$Z_{2^k}=\{g^{ri}|i\in\{0,1,\dots,2^k-1\}\}$，其生成元的阶为$2^k$，于是我们可取$\omega_n=g^r$这里$g$为模$p$意义下的原根。因此对于素数$p=2^kr+1$，我们可在模$p$意义下进行长度为$2^k$或其因子的FFT。

```
namespace NTT1 {

using ::mul;
using ::inv;

const int W = 18, S = 1 << W, g = 3;
int w[S + 1], rev[S << 1], *r[W + 1];
void init() {
    for (int s = 0; s <= W&&(r[s]=rev+(1<<s),1); ++s)
        for (int i = 0; i != (1 << s); ++i)
            r[s][i] = (r[s][i >> 1] >> 1) | ((i & 1) << (s - 1));
    w[0] = 1; w[1] = qpm(g, (P - 1) / S);
    for (int i = 2; i <= S; ++i) w[i] = mul(w[i - 1], w[1]);
}

int m, s, im;
int init(int n) {
    for (s = 0, m = 1; m < n; m <<= 1, ++s);
    im = inv(m); return m;
}

void ntt(int* p, int t) {
    for (int i = 0; i != m; ++i) if (i < r[s][i]) swap(p[i], p[r[s][i]]);
    for (int i = 1, z = 0; i != m; i <<= 1, z++)
        for (int j = 0; j != m; j += (i << 1))
            for (int k = 0, u, v; k != i; k++)
                u = p[j+k], v = mul(w[(t?(i<<1)-k:k)<<W-z-1], p[i+j+k]),
                p[j + k] = add(u, v), p[i + j + k] = sub(u, v);
    if (t) for (int i = 0; i != m; ++i) p[i] = mul(p[i], im);
}

int px[S], py[S];
vi pmul(const vi& p1, const vi& p2, int n = 0) {
    int n1 = p1.size(), n2 = p2.size(), n3 = n1 + n2 - 1;
    init(n3);
    copy_n(p1.begin(), n1, px); fill(px + n1, px + m, 0);
    copy_n(p2.begin(), n2, py); fill(py + n2, py + m, 0);
    ntt(px, 0); ntt(py, 0);
    for (int i = 0; i != m; ++i) px[i] = mul(px[i], py[i]);
    ntt(px, 1); vi p3(n3); copy_n(px, n3, p3.begin());
    if (n && n3 > n) p3.resize(n); return p3;
}

vi pinv(const vi& p1) {
    int n1 = p1.size(), n2 = (n1 + 1) >> 1;
    if (n1 == 1) return { inv(p1[0]) };
    else {
        vi p2 = pinv(vi(p1.begin(), p1.begin() + n2));
        init(n1 << 1);
        copy_n(p1.begin(), n1, px); fill(px + n1, px + m, 0);
        copy_n(p2.begin(), n2, py); fill(py + n2, py + m, 0);
        ntt(px, 0); ntt(py, 0);
```

```
        for (int i = 0; i != m; ++i)
            px[i] = mul(sub(2, mul(px[i], py[i])), py[i]);
        ntt(px, 1); vi p3(n1); copy_n(px, n1, p3.begin());
        return p3;
    }
}

}

using NTT1::init;
using NTT1::pmul;
using NTT1::pinv;
```

任意模数NTT

相当于9次普通NTT，巨大常数，$O(n \log n)$

碰到了建议重新思考生成函数之外的做法或弃题（迫真）。

```
namespace NTT2 {

    typedef array<int, 3> mint;
    const int M[3] = { 167772161, 469762049, 998244353 }, g = 3;

    template<const int i> int add(int a, int b) { int r = a + b; return r < M[i] ? r : r - M[i]; }
    template<const int i> int sub(int a, int b) { int r = a - b; return r < 0 ? r + M[i] : r; }
    template<const int i> int mul(long long a, long long b) { return a * b % M[i]; }
    template<const int i> int inv(int a) { return a == 1 ? a : mul<i>(inv<i>(M[i] % a), M[i] - M[i] / a); }
    template<const int i> int qpm(int a, int b, int r = 1) {
        do if (b & 1) r = mul<i>(r, a); while (a = mul<i>(a, a), b >>= 1);
        return r;
    }

    inline mint mul(mint a, mint b) { return { mul<0>(a[0], b[0]), mul<1>(a[1], b[1]), mul<2>(a[2], b[2]) }; }

    inline int crt(const mint& x){
        static const int o0 = inv<1>(M[0]);
        static const int o1 = inv<2>(1ll * M[0] * M[1] % M[2]);
        long long a = 1ll * (x[1] - x[0] + M[1]) * o0 % M[1] * M[0] + x[0];
        return (1ll * (x[2] - a % M[2] + M[2]) * o1 % M[2] * M[0] % P * M[1] + a) % P;
    }

    const int W = 18, S = 1 << W;
    int rev[S << 1], *r[W + 1];
    mint w[S + 1];
    void init() {
        for (int s = 0; s <= W&&(r[s]=rev+(1<<s),1); ++s)
            for (int i = 0; i != (1 << s); ++i)
                r[s][i] = (r[s][i >> 1] >> 1) | ((i & 1) << (s - 1));
        w[0] = { 1, 1, 1 };
        w[1][0] = qpm<0>(g, (M[0] - 1) / S);
```

```
        w[1][1] = qpm<1>(g, (M[1] - 1) / S);
        w[1][2] = qpm<2>(g, (M[2] - 1) / S);
        for (int i = 2; i <= S; ++i) w[i] = mul(w[i - 1], w[1]);
    }

    int m, s; mint im;
    void init(int n) {
        for (s = 0, m = 1; m < n; m <<= 1, ++s);
        im = { inv<0>(m), inv<1>(m), inv<2>(m) };
    }

    void ntt(mint* p, int t) {
        for (int i = 0; i != m; ++i) if (i < r[s][i]) swap(p[i], p[r[s]
[i]]);
        for (int i = 1, z = 0; i != m; i <<= 1, z++)
            for (int j = 0; j != m; j += (i << 1))
                for (int k = 0; k != i; k++) {
                    mint u=p[j+k], v=mul(w[(t?(i<<1)-k:k)<<W-z-1],
p[i+j+k]);
                    p[j + k] = { add<0>(u[0], v[0]), add<1>(u[1], v[1]),
add<2>(u[2], v[2]) };
                    p[i + j + k] = { sub<0>(u[0], v[0]), sub<1>(u[1], v[1]),
sub<2>(u[2], v[2]) };
                }
        if (t) for (int i = 0; i != m; ++i) p[i] = mul(p[i], im);
    }

    mint px[S], py[S];
    vi pmul(const vi& p1, const vi& p2, int n = 0) {
        int n1 = p1.size(), n2 = p2.size(), n3 = n1 + n2 - 1;
        init(n3);
        for (int i = 0; i != n1; ++i) px[i] = { p1[i] % M[0], p1[i] % M[1],
p1[i] % M[2] };
        for (int i = n1; i != m; ++i) px[i] = { 0, 0, 0 };
        for (int i = 0; i != n2; ++i) py[i] = { p2[i] % M[0], p2[i] % M[1],
p2[i] % M[2] };
        for (int i = n2; i != m; ++i) py[i] = { 0, 0, 0 };
        ntt(px, 0); ntt(py, 0);
        for (int i = 0; i != m; ++i) px[i] = mul(px[i], py[i]);
        ntt(px, 1);
        vi p3(n3);
        for (int i = 0; i != n3; ++i) p3[i] = crt(px[i]);
        if (n && n3 > n) p3.resize(n); return p3;
    }

    vi pinv(const vi& p1) {
        int n1 = p1.size(), n2 = (n1 + 1) >> 1;
        if (n1 == 1) return { ::inv(p1[0]) };
        vi p2 = pinv(vi(p1.begin(), p1.begin() + n2));
        return pmul(psub({2}, pmul(p1, p2)), p2, n1);
    }
}

using NTT2::init;
using NTT2::pmul;
using NTT2::pinv;
```

多项式操作

卷积

给定多项式$f(x),g(x)$，求$h(x)=f(x)g(x)$。

对$f(x)$与$g(x)$分别进行FFT/NTT，计算逐点乘积后进行逆变换即可。

在n小的时候暴力计算可减小常数。

```
int px[S], py[S];
vi mul(const vi& p1, const vi& p2, int n = 0) {
    int n1 = p1.size(), n2 = p2.size(), n3 = n1 + n2 - 1;
    init(n3);
    copy_n(p1.begin(), n1, px); fill(px + n1, px + m, 0);
    copy_n(p2.begin(), n2, py); fill(py + n2, py + m, 0);
    ntt(px, 0); ntt(py, 0);
    for (int i = 0; i != m; ++i) px[i] = mul(px[i], py[i]);
    ntt(px, 1); vi p3(n3); copy_n(px, n3, p3.begin());
    if (n && n3 > n) p3.resize(n); return p3;
}
```

分治卷积

给定多项式$f(x)$，求$g(x)$满足$g_i=\sum_{j=0}^{i-1}f_{i-j}g_j$，其中$g_0$已给定。

使用分治计算$g_{l\dots r-1}$。

设$m=\lceil (l+r)/2\rceil,w=m-l$。

对左半边分治下去，即计算$g_{l\dots m-1}$。

考虑计算左边对右边的贡献（$i\geq m$）：

$$\sum_{j=l}^{i-1}f_{i-j}g_j=\sum_{j=l}^{m-1}f_{i-j}g_j+\sum_{j=m}^{i-1}f_{i-j}g_j$$

考虑用卷积计算第一项。令

$$h_j=\begin{cases}g_{l+j} & j<w\\ 0 & j\geq w\end{cases}$$

$$(h*g)_{w+i}=\sum_{j=0}^{w+i}h_jg_{w+i-j}=\sum_{j=0}^{w-1}h_jg_{w+i-j}=\sum_{j=0}^{w-1}g_{l+j}f_{w+i-j}=\sum_{j=l}^{m-1}f_{m+i-j}g_j$$

即第一项可以通过计算$\{g_l,g_{l+1},\cdots,g_{m-1}\}$与$f$的卷积得到。

时间复杂度$O(n\log^2 n)$

```cpp
void conv(const vi& f, vi& g, int l, int r) {
    if (l + 1 == r) return;
    else {
        int m = (l + r) >> 1;
        conv(f, g, l, m);
        vi h = mul(
            vi(g.begin() + l, g.begin() + m),
            vi(f.begin(), f.begin() + r - l + 1)
        );
        for (int i = m; i != r; ++i) f[i] = add(f[i], h[i - l]);
        conv(f, g, m, r);
    }
}
```

有依赖的分治卷积

给定运算$\bigoplus$，函数t与递推初值g_1和递推关系

$$f_i = \bigoplus_{j=1}^{i} g_j, g_i = t\left(\sum_{j=1}^{i-1} f_{i-j} g_j\right)$$

使用分治计算$f_{l\dots r-1}$和$g_{l\dots r-1}$。且后者中不涉及区间$l\dots r-1$的贡献已经计算完成。

设$m = \lceil (l+r)/2 \rceil, w = m - l$。

对左半边分治下去，即计算$g_{l\dots m-1}$。

考虑计算左边对右边的贡献$(i \geq m)$，即计算：

$$\sum_{(l\leq j_1 \vee l\leq j_2)\wedge j_1<m\wedge j_2<m\wedge j_1+j_2=i} f_{j_1} g_{j_2}$$

当$l = 1$时，该式等价于$u = \{f_1, f_2, \dots, f_{m-1}\}$与$v = \{g_1, g_2, \dots, g_{m-1}\}$的卷积的第$i$项。

当$l \neq 1$时，根据分治的性质，当$l \neq 1$时有$j_1 + j_2 \geq 2l \geq r > i$，因此$l \leq j_1$ 与 $l \leq j_2$不能同时满足。

分类讨论得该式等于

$$\sum_{j_1=l}^{m-1} f_{j_1} g_{i-j_1} + \sum_{j_2=l}^{m-1} f_{i-j_2} g_{j_2}$$

第一项等价于$u = \{f_l, f_{l+1}, \cdots, f_{m-1}\}$与$v = \{g_1, g_2, \cdots, g_{r-l+1}\}$的卷积的第$i-l-1$项。

第二项等价于$u = \{f_1, f_2, \cdots, f_{r-l+1}\}$与$v = \{g_l, g_{l+1}, \cdots, g_{m-1}\}$的卷积的第$i-l-1$项。

时间复杂度$O(n \log^2 n)$

```cpp
void conv(vi& f, vi& g, int l, int r) {
    if (l + 1 == r) {
        //  g[l]=t(g[l])
        //  Calculate f[l]
    }
    else {
        int m = (l + r + 1) >> 1;
        conv(f, g, l, m);
        if (l == 1) {
            vi h = pmul(
                vi(f.begin() + 1, f.begin() + m),
```

```
                vi(g.begin() + 1, g.begin() + m)
            );
            for (int i = m; i != r; ++i) g[i] = add(g[i], h[i - l - 1]);
        }
        else {
            assert(m - l + 1 <= l);
            vi h1 = pmul(
                vi(f.begin() + l, f.begin() + m),
                vi(g.begin() + 1, g.begin() + r - l + 1)
            ),
               h2 = pmul(
                vi(f.begin() + 1, f.begin() + r - l + 1),
                vi(g.begin() + l, g.begin() + m)
            );
            for (int i = m; i != r; ++i) g[i] = add(g[i], add(h1[i - l - 1],
h2[i - l - 1]));
        }
        conv(f, g, m, r);
    }
}
```

缩放

给定$f(x)$，求$g(x)=f(kx)$

$$g(x)=f(kx)=\sum_{i=0}^{n-1}a_ik^ix^i$$

```
vi scale(const vi& a, ll d) {
    vi b = a;
    for (int i = 0; i != b.size(); ++i)
        b[i] = mul(b[i], qpm(d, i));
    return b;
}
```

平移

前置：卷积

给定$f(x)$，求$g(x)=f(x+\delta)$。

$$f(x+\delta)=\sum_{i=0}^{n-1}a_i\sum_{j=0}^{i}\binom{i}{j}x^j\delta^{i-j}=\sum_{j=0}^{n-1}x^j\sum_{i=j}^{n-1}\binom{i}{j}a_i\delta^{i-j}$$

$$=\sum_{j=0}^{n-1}x^j\sum_{i=0}^{n-j-1}\binom{n-i-1}{j}a_{n-i-1}\delta^{n-i-j-1}=\sum_{j=0}^{n-1}x^{n-j-1}\sum_{i=0}^{j}\binom{n-i-1}{n-j-1}a_{n-i-1}\delta^{j-i}$$

令$c_i=a_{n-i-1}(n-i-1)!$，$d_i=\frac{\delta^i}{i!}$

$$=\sum_{j=0}^{n}\frac{x^{n-j}}{(n-j)!}\sum_{i=0}^{j}\frac{(n-i)!}{(j-i)!}a_{n-i}\delta^{j-i}=\sum_{j=0}^{n}\frac{x^{n-j}}{(n-j)!}\sum_{i=0}^{j}c_id_{j-i}$$

将序列c与序列d卷起来再乘以阶乘即可。

时间复杂度$O(n\log n)$，大常数。

```
vi shift(const vi& a, ll d) {
    int n = a.size();
    vi b = a, c(n);
    reverse(b.begin(), b.end());
    for (int i = 0; i != n; ++i) {
        b[i] = mul(b[i], fac[n - i - 1]);
        if (!i) c[i] = 1;
        else c[i] = mul(c[i - 1], mul(d, invs[i]));
    }
    vi r = mul(b, c, n);
    reverse(r.begin(), r.end());
    return egf(r);
}
```

求导

给定$f(x)$，求$g(x)=f'(x)$。

$$g(x)=\sum_{i=0}^{n-2}(i+1)a_{i+1}x^i$$

时间复杂度$O(n)$。

```
vi deriv(const vi& p1) {
    int n1 = p1.size();
    vi p2(n1 - 1);
    for (int i = 1; i != n1; ++i) p2[i - 1] = mul(i, p1[i]);
    return p2;
}
```

积分

给定$f(x)=\sum_{i=0}^{n-1}a_ix^i$，求$g(x)=\int_0^x f(t)dt$。

$$g(x)=\sum_{i=1}^{n}\frac{a_{i-1}}{i}x^i$$

时间复杂度$O(n)$。

```
vi integ(const vi& p1) {
    int n1 = p1.size();
    vi p2(n1 + 1, 0);
    for (int i = 0; i != n1; ++i) p2[i + 1] = mul(p1[i], invs[i + 1]);
    return p2;
}
```

牛顿迭代

注：以下涉及到牛顿迭代过程中的$\frac{n}{2}$均自动向上取整。

给定多项式$t(g)$，求$g(x)$使得$t(g)\equiv 0 \mod x^n$。

设已求出$h(x)$使得$t(h)\equiv 0 \mod x^{\frac{n}{2}}$。

因为有$t(g)\equiv 0 \mod x^n$，所以有$t(g)\equiv 0 \mod x^{\frac{n}{2}}$，因此$g(x)\equiv h(x) \mod x^{\frac{n}{2}}$。

考虑$t(g)$在h处的泰勒展开

$$t(g) \equiv \sum_{k=0}^{\infty} \frac{t^{(k)}(h)}{k!}(g-h)^k \mod x^n$$

因为$g(x)-h(x) \equiv 0 \mod x^{\frac{n}{2}}$，所以第二项往后的项全部为0，我们得到

$$0 \equiv t(g) \equiv t(h) + t'(h)(g-h) \mod x^n$$

解得

$$g \equiv h - \frac{t(h)}{t'(h)} \mod x^n$$

边界：当$n=1$时根据具体情况处理。

涉及到牛顿迭代的多项式算法基本都自带巨大常数，$n\log n$跑$n=10^5$要半秒左右。

求逆

前置：卷积

给定$f(x)$，求$g(x)$使得$f(x)g(x) \equiv 1 \mod x^n$。

原始形式：

$$g_0 = f_0^{-1}, \sum_{j=0}^{i} f_j g_{i-j} = 0$$

即

$$g_i = \begin{cases} f_0^{-1} & i=0 \\ -f_0^{-1}\sum_{j=1}^{i} f_j g_{i-j} & i \geq 1 \end{cases}$$

考虑牛顿迭代：设$t(g) = \frac{1}{g} - f$，则迭代方程为

$$g(x) = h(x) - \frac{\frac{1}{h(x)} - f(x)}{-\frac{1}{h^2(x)}} = 2h(x) - h^2(x)f(x) \mod x^n$$

边界：当$n=1$时直接求常数的乘法逆即可。

时间复杂度$O(n\log n)$，大常数。

```
vi inv(const vi& p1) {
    int n1 = p1.size(), n2 = (n1 + 1) >> 1;
    if (n1 == 1) return { inv(p1[0]) };
    else {
        vi p2 = inv(vi(p1.begin(), p1.begin() + n2));
        init(n1 << 1);
        copy_n(p1.begin(), n1, px); fill(px + n1, px + m, 0);
        copy_n(p2.begin(), n2, py); fill(py + n2, py + m, 0);
        ntt(px, 0); ntt(py, 0);
        for (int i = 0; i != m; ++i)
            px[i] = mul(sub(2, mul(px[i], py[i])), py[i]);
        ntt(px, 1); vi p3(n1); copy_n(px, n1, p3.begin());
        return p3;
    }
}
```

除法与取模

前置：求逆

给定$f(x)$与$g(x)$，求$q(x)$与$r(x)$使得$f(x)=q(x)g(x)+r(x)$。

其中$q(x)$为$n-m+1$次多项式，$r(x)$为$m-1$次多项式。

原始形式：直接模拟$O(n^2)$的多项式除法/取模

```
pair<vi, vi> div(const vi& p1, const vi& p2) {
    int n1 = p1.size(), n2 = p2.size();
    if (n1 < n2) return { vi(), p1 };
    vi p3(n1 - n2 + 1, 0), p4 = p1;
    for (int i = n1 - 1; i >= n2 - 1; --i) {
        p3[i - n2 + 1] = mul(p4[i], inv(p2[n2 - 1]));
        for (int j = 0; j != n2; ++j)
            p4[i - n2 + 1 + j] = sub(p4[i - n2 + 1 + j], mul(p2[j], p3[i -
n2 + 1]));
    }
    while (!p4.empty() && p4.back() == 0) p4.pop_back();
    return { p3, p4 };
}
```

考虑将$f(x)$的系数前后翻转

$$f_R(x)=x^{n-1}f(\frac{1}{x})=\sum_{i=0}^{n-1}a_ix^{n-i-1}$$

将$\frac{1}{x}$作为参数带入上式并两边乘上x^{n-1}可得

$$x^{n-1}f(\frac{1}{x})=x^{n-m+1}q(\frac{1}{x})x^{m-1}g(\frac{1}{x})+x^{n-m+1}x^{m-1}r(\frac{1}{x})$$

$$f_R(x)=q_R(x)g_R(x)+x^{n-m+1}r_R(x)$$

$$f_R(x)=q_R(x)g_R(x)\mod x^{n-m+1}$$

$$q_R(x)=f_R(x)g_R^{-1}(x)\mod x^{n-m+1}$$

对g_R多项式求逆后卷上$f_R(x)$并抛掉多余系数即可得到$q_R(x)$，进一步可求出$r_R(x)$。

时间复杂度$O(n\log n)$，大常数。

```
pair<vi, vi> div(const vi& p1, const vi& p2) {
    int n1 = p1.size(), n2 = p2.size(), n3 = n1 - n2 + 1;
    if (n3 <= 0) return { { 0 }, p1 };
    vi p1r = p1, p2r = p2;
    reverse(p1r.begin(), p1r.end());
    reverse(p2r.begin(), p2r.end());
    p1r.resize(n3, 0); p2r.resize(n3, 0);
    vi p3 = mul(p1r, inv(p2r), n3);
    reverse(p3.begin(), p3.end());
    vi p4 = sub(p1, mul(p2, p3));
    p4.resize(n2 - 1, 0);
    return { p3, p4 };
}
```

注：`first`是$q(x)$，`second`是$r(x)$。

开根号

前置：卷积，求逆

给定$f(x)$，求$g(x)$使得$g(x)^2 \equiv f(x) \mod x^n$。

考虑牛顿迭代：设$t(g) = g^2 - f$，则迭代方程为

$$g(x) = h(x) - \frac{h^2(x) - f(x)}{2h(x)} = \frac{h^2(x) + f(x)}{2h(x)} \mod x^n$$

边界：当$n = 1$时直接求常数的模意义下二次剩余即可。

时间复杂度$O(n \log n)$，大常数。

```
int msqrt(int n) {
    if (!n) return 0;
    int q = P - 1, s = 0, z = 2;
    //while (~q & 1) q >>= 1, s++;
    q >>= (s = __builtin_ctzll(q));
    if (s == 1) return qpm(n, (P + 1) / 4);
    while(qpm(z, (P - 1) / 2) == 1) ++z;
    int c = qpm(z, q), t = qpm(n, q),
       r = qpm(n, (q + 1) / 2), m = s;
    while(t % P != 1) {
        ll i = 1; while(qpm(t, 1ll << i) != 1) ++i;
        ll b = qpm(c, 1ll << (m - i - 1));
        r = mul(r, b); c = mul(b, b);
        t = mul(t, c); m = i;
    }
    return min(r, P - r);
}

vi sqrt(const vi& p1) {
    int n1 = p1.size(), n2 = (n1 + 1) >> 1;
    if (n1 == 1) return { ::msqrt(p1[0]) };
    else {
        vi p2 = sqrt(vi(p1.begin(), p1.begin() + n2));
        vi p3 = mul(p2, 2); p3.resize(n1); p3 = inv(p3);
        return mul(add(mul(p2, p2, n1), p1), p3, n1);
    }
}
```

对数

前置：求逆，求导，积分

给定$f(x)$，求$g(x) = \ln f(x) \mod x^n$。

原始形式：

$$\ln f(x) = \sum_{j=1}^{+\infty} (-1)^{j+1} \frac{f(x)^j}{j}$$

$$g(x) = \ln f(x) = \int_0^x \frac{f'(t)}{f(t)} dt$$

时间复杂度$O(n\log n)$，大常数。

```
vi log(const vi& p1) {
    return integ(mul(deriv(p1), inv(p1), p1.size() - 1));
}
```

指数

前置：卷积，对数

给定$f(x)$，求$g(x)=\exp f(x) \mod x^n$。其中$f(0)=a_0=0$。

原始形式：

$$\exp f(x)=\sum_{i=0}^{\infty}\frac{f(x)^i}{i!}$$

$$b_i=\sum$$

考虑牛顿迭代：设$t(g)=\ln g-f$，则迭代方程为

$$g(x)=h(x)-\frac{\ln h-x}{\frac{1}{h}}=h(x)(1-\ln h+f) \mod x^n$$

边界：当$n=1$时返回常数1。

时间复杂度$O(n\log n)$，大常数。

```
vi exp(const vi& p1) {
    if (p1.size() == 1) return { 1 };
    else {
        vi p2 = exp({p1.begin(),p1.begin()+(p1.size()+1>>1)});
        p2.resize(p1.size(), 0);
        return mul(p2, add(sub({ 1 }, log(p2)), p1), p1.size());
    }
}
```

快速幂

前置：指数

给定$f(x)$，求$g(x)=f(x)^k$。

注意到$g(x)=f(x)^k=\exp\ln f(x)^k=\exp k\ln f(x)$，因此快速幂可在$O(n\log n)$内完成。

若$a_0\neq 1$，则在取对数前需进行如下特判：

若$a_i,i\in\{0,1,\dots,j-1\}$均为0，则可除以一个$x^j$后将$g$像后移$nj$位。

若进行上一步后仍有$a_0\neq 1$，则将整个序列除以a_0，后将整个序列乘以a_0^k。

若nk较小则在NTT后直接对点值表示快速幂再逆NTT即可(CF1096G)。

时间复杂度$O(n\log n)$，巨大常数。

```
vi pow(const vi& p1, int k) {
    int n1 = p1.size(), n2 = n1;
    while (n2 && !p1[n1 - n2]) n2--;
    int n3 = max(n1 - 1ll * (n1 - n2) * k, 0ll);
    if (!n2 || !n3) return vi(n1, 0);
    vi p2(p1.begin() + n1 - n2, p1.begin() + n1 - n2 + n3);
    ll c = p2[0]; p2 = mul(exp(mul(log(mul(p2, ::inv(c))), k)), qpm(c, k));
    p2.resize(n1, 0); rotate(p2.begin(), p2.begin() + n3, p2.end());
    return p2;
}
```

欧拉变换

前置：指数

给定$f(x)$，求$\mathscr{E}(f)(x)=\prod_{i=1}^{+\infty}\left(1-x^i\right)^{-a_i}$

用exp展开后展开$-\ln(1-x^i)$后可得

$$\mathscr{E}(f)(x)=\exp\left(\sum_{j=1}^{+\infty}\frac{f(x^j)}{j}\right)$$

exp内的多项式可在$O(n\log n)$内求出。

注：可扩展到形如$(\prod_{i=1}^{+\infty}\left(1-b_ix^i\right)^{a_i})$的多项式。

时间复杂度$O(n\log n)$。

```
vi eul(const vi& p1) {
    int n = p1.size();
    vi p2 = p1;
    for (int j = 2; j != n; ++j)
        for (int i = 1; i * j < n; ++i)
            p2[i * j] = add(p2[i * j], mul(p1[i], invs[j]));
    return exp(p2);
}
```

多点求值

前置：除法

给定$f(x)$，和$x_i,i\in\{1,2,\ldots,m\}$，求$f(x_i),i\in\{1,2,\ldots,m\}$

构造多项式$g(x)=\prod_{i=1}^{m}(x-x_i)$，注意到$g(x_i)=0$，考虑多项式除法

$$f(x_i)=q(x_i)g(x_i)+r(x_i)=r(x_i)$$

对g取模后只需对r求值即可。

$g_{l,r}=\prod_{i=l}^{r}(x-x_i)$可分治求出，将中间结果保存至线段树上后再进行分治，即将对f求其在$x_i,i\in\{l,\ldots,r\}$上的值分治为求$f\mod g_{l,mid-1}$在$x_i,i\in\{l,\ldots,mid-1\}$上的值和求$f\mod g_{mid,r}$在$x_i,i\in\{mid,\ldots,r\}$上的值。因NTT常数巨大，所以在分治到一定程度时直接转秦九韶暴力求值可降低常数。

时间复杂度$O(n\log^2 n)$，巨大常数。

```
vi est[N];

```

```
int eval(const vi& p, int x) {
    int r = 0;
    for (int i = (int)p.size() - 1; i >= 0; --i)
        r = add(p[i], mul(r, x));
    return r;
}

void eval0(const vi& x, int p, int lb, int rb) {
    if (lb + 1 == rb) est[p] = { sub(0, x[lb]), 1 };
    else {
        int mid = (lb + rb) >> 1;
        eval0(x, p << 1, lb, mid);
        eval0(x, p << 1 | 1, mid, rb);
        est[p] = mul(est[p << 1], est[p << 1 | 1]);
    }
}

void eval1(const vi& r, const vi& x,
           int p, int lb, int rb) {
    vi w = div(r, est[p]).second;
    if (lb + 100 >= rb)
        for (int i = lb; i != rb; ++i)
            est[0][i] = eval(w, x[i]);
    else {
        int mid = (lb + rb) >> 1;
        eval1(w, x, p << 1, lb, mid);
        eval1(w, x, p << 1 | 1, mid, rb);
    }
}

vi eval(const vi& p, const vi& x) {
    eval0(x, 1, 0, x.size());
    est[0].resize(x.size());
    eval1(p, x, 1, 0, x.size());
    return est[0];
}
```

快速连续插值

前置：卷积

给定$f(0), f(1), \ldots, f(n)$，求$f(m), f(m+1), \ldots, f(m+n)$模意义下的值。其中$m > n$。

由插值公式可得

$$f(m+x) = \sum_{i=0}^{n} f(i) \prod_{j \neq i} \frac{m+x-j}{i-j} = \sum_{i=0}^{n} f(i) \frac{(m+x)!/(m+x-n-1)!}{(m+x-i)(-1)^{n-i} i!(n-i)!}$$

令

$$u_i = \frac{f(i)}{(-1)^{n-i} i!(n-i)!}$$

当$i > n$时$u_i = 0$

$$v_i = \frac{1}{m-n+i}$$

可得

$$(u*v)_x=\sum_{i=0}^{x}\frac{f(i)}{(-1)^{n-i}i!(n-i)!(m-n+x-i)}$$

$$(u*v)_{n+x}=\sum_{i=0}^{n}\frac{f(i)}{(-1)^{n-i}i!(n-i)!(m+x-i)}$$

即

$$f(m+x)=(u*v)_{n+x}\prod_{i=m+x-n}^{m+x}i$$

于是可用一次(任意模数)NTT求出$f(m),f(m+1),\ldots,f(m+n)$。

总复杂度$O(n\log n)$。

```
vi lintp(const vi& y, int m) {
    int n = y.size() - 1; vi z1(y), z2(2 * n + 1, 0);
    for (int i = 0; i <= 2 * n; ++i) z2[i] = inv(m - n + i);
    for (int i = 0; i <= n; ++i) {
        int c = mul(ifac[n - i], ifac[i]);
        if ((n - i) & 1) c = sub(0, c);
        z1[i] = mul(z1[i], c);
    }
    z1 = mul(z1, z2);
    z1.erase(z1.begin(), z1.begin() + n);
    z1.resize(n + 1);
    int f = 1;
    for (int i = m - n - 1; i <= m - 1; ++i) if (i) f = mul(f, i);
    for (int i = 0; i <= n; ++i) {
        f = mul(f, m + i);
        if (m - n - 1 + i) f = mul(f, inv(m - n - 1 + i));
        z1[i] = mul(z1[i], f);
    }
    return z1;
}
```

*快速插值

拉格朗日反演

给定多项式$F(x)$，若存在$F^{-1}(x)$满足$F^{-1}(F(x))=x$，则称F^{-1}与F互为复合逆。

给定F,H，则有

$$[x^n]H(F^{-1}(x))=\frac{1}{n}[x^{n-1}]H'(x)\left(\frac{x}{F(x)}\right)^n$$

特别的，当$H(x)=x$时，有

$$[x^n]F^{-1}(x)=\frac{1}{n}[x^{n-1}]\left(\frac{x}{F(x)}\right)^n$$

计数序列

等比数列与求和

当$q \neq 1$时有

$$S_0(n) = \sum_{i=0}^{n} q^i = \frac{q^{n+1}-1}{q-1}$$

$$S_k(n) = \sum_{i=0}^{n} i^k q^i = \frac{1}{q-1}\left[(n+1)^k q^{n+1} - q\sum_{j=0}^{k-1}\binom{k}{j}S_j(n)\right]$$

证明：扰动法。

$$S_k(n) = 0^k q^0 + \sum_{i=1}^{n} i^k q^i = S_k(n-1) + n^k q^n$$

$$n^k q^n = \sum_{i=0}^{n-1}(i+1)^k q^{i+1} - S_k(n-1) = \sum_{i=0}^{n-1}\sum_{j=0}^{k}\binom{k}{j} i^j q^{i+1} - S_k(n-1)$$

$$= q\sum_{j=0}^{k}\binom{k}{j}S_j(n-1) - S_k(n-1)$$

因此有

$$q\sum_{j=0}^{k-1}\binom{k}{j}S_j(n-1) + (q-1)S_k(n-1) = n^k q_n$$

当$q = 1$时退化为等幂求和。

二项式系数

定义：

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

组合意义：

只能走$(1,0)$和$(0,1)$，从$(0,0)$走到(n,m)的路径数等于$\binom{n+m}{n}$。

证明：对于每条路径，将每一步映射成`U`和`R`后唯一对应一个恰好有n个`U`和m个`R`的字符串，即从$n+m$个位置中钦定n个位置是`U`。

命题1：

$$\binom{n}{k} = \binom{n}{n-k}$$

证明：显然。

组合意义：钦定n个`U`和m个`R`是等价的。

命题2：

$$\binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1} = \frac{n}{n-k}\binom{n-1}{k} = \frac{n}{k}\binom{n-1}{k-1}$$

$$= \frac{k+1}{n-k}\binom{n}{k+1} = \frac{n+1-k}{n+1}\binom{n+1}{k} = \frac{k+1}{n+1}\binom{n+1}{k+1}$$

证明：按定义展开即可。

命题3：

$$\binom{n}{k}=\binom{n-1}{k-1}+\binom{n-1}{k}$$

证明：

$$\binom{n-1}{k-1}+\binom{n-1}{k}=\frac{k}{n}\binom{n}{k}+\frac{n-k}{n}\binom{n}{k}=\binom{n}{k}$$

组合意义：$(0,0)-(n,m)$的方案数等于$(0,0)-(n-1,m)$的方案数加上$(0,0)-(n,m-1)$的方案数。

命题4：

$$\sum_{i=0}^{k}\binom{n+i}{n}=\sum_{i=0}^{k}\binom{n+i}{i}=\binom{n+k+1}{k}=\binom{n+k+1}{n+1}$$

证明：左右按命题1显然，中间的可以归纳。当$k=1$时成立。当$k\geq 2$时有

$$\sum_{i=0}^{k}\binom{n+i}{i}+\binom{n+k+1}{k+1}=\binom{n+k+2}{k+1}=\binom{n+(k+1)+1}{(k+1)}$$

命题5(Vandermonde卷积)：设$k\leq n\leq m$，则

$$\sum_{i=0}^{k}\binom{n}{i}\binom{m}{k-i}=\binom{n+m}{k}$$

组合意义：$(0,0)-(n,m)$的方案数等于所有形如$(0,0)-(k-i,i)-(n,m)$的路径数之和。

证明：任意$(0,0)-(n,m)$的路径都恰好有一个点在直线$x+y=k$上。枚举这个点，前半部分的方案数是

$\binom{k}{i}$，后半部分的方案数是$\binom{n+m-k}{m-i}$。做代换即可。

命题6：

$$\sum_{i=0}^{m}\binom{n_1+i}{n_1}\binom{n_2+m-i}{n_2}=\binom{n_1+n_2+1+m}{m}$$

组合意义：$(0,0)-(n_1+n_2+1,m)$的方案数等于从点(n_1,i)往右走的路径数之和。

证明：按定义即可。

错排

定义（错排数）：长度为n的不存在i使得$p_i=i$的排列数为D_n

考虑容斥：设$f(S)$为S中数满足$p_i=i$且其他数都不满足的方案数，设$g(S)$为S中数满足$p_i=i$且其他数可以满足也可以不满足的方案数，则

$$g(S)=(n-|S|)!=\sum_{S\subseteq T}f(S)$$

$$f(S)=\sum_{S\subseteq T}(-1)^{|T|-|S|}g(T)=\sum_{S\subseteq T}(-1)^{|T|-|S|}(n-|T|)!$$

$$D_n=f(\emptyset)=\sum_{k=|T|=0}^{n}(-1)^k\binom{n}{k}(n-k)!=n!\sum_{k=0}^{n}(-1)^k\frac{1}{k!}$$

考虑D_n的EGF：

$$F(x)=\sum_{n=0}x^n\sum_{k=0}^{n}\frac{(-1)^k}{k!}=\sum_{k=0}\frac{(-1)^k}{k!}\sum_{n=k}x^n=\frac{1}{1-x}\sum_{k=0}\frac{(-x)^k}{k!}=\frac{\exp(-x)}{1-x}$$

可通过$x\frac{\partial}{\partial x}\log$求出其递推式：

$$x\frac{\partial}{\partial x}\log F(x)=x\frac{F'(x)}{F(x)}$$

$$x\frac{\partial}{\partial x}\log\left\{\frac{\exp(-x)}{1-x}\right\}=x\frac{\partial}{\partial x}(-x-\log(1-x))=\frac{x^2}{1-x}=x^2+x^3+\cdots$$

因此有

$$(1-x)F'(x)=xF(x)$$

$$(n+1)\frac{D_{n+1}}{(n+1)!}-n\frac{D_n}{n!}=\frac{D_{n-1}}{(n-1)!}$$

$$D_{n+1}=n(D_n+D_{n-1})$$

Fibonacci数

定义：$F_0=0,F_1=1,F_n=F_{n-1}+F_{n-2}$

设转移矩阵$A=\begin{bmatrix}1&1\\1&0\end{bmatrix}$，则$\begin{bmatrix}F_n\\F_{n-1}\end{bmatrix}=A^{n-1}\begin{bmatrix}F_1\\F_0\end{bmatrix}$。

且因$A=\begin{bmatrix}F_2&F_1\\F_1&F_0\end{bmatrix}$有$A^n=\begin{bmatrix}F_{n+1}&F_n\\F_n&F_{n-1}\end{bmatrix}$。

性质：$\sum_{k=1}^{n}F_k=F_{n+2}-1$

证明：$\sum_{k=1}^{n+1}F_k=\sum_{k=1}^{n}F_k+F_{n+1}=F_{n+2}-1+F_{n+1}=F_{n+3}-1$

性质：$\sum_{k=1}^{n}F_k^2=F_nF_{n+1}$

证明：$\sum_{k=1}^{n+1}=\sum_{k=1}^{n}F_k^2+F_{n+1}^2=F_nF_{n+1}+F_{n+1}^2=F_{n+1}(F_n+F_{n+1})=F_{n+1}F_{n+2}$

性质：$\sum_{k=0}^{n-2}F_kF_{k+3}=F_n^2-1$

证明：扰动平方和

$$\sum_{k=1}^{n}F_k^2=1+\sum_{k=2}^{n}F_k^2=\sum_{k=1}^{n-1}F_k^2+F_n^2$$

$$\sum_{k=1}^{n-1}(F_{k+1}^2-F_k^2)=F_n^2-1$$

$$\sum_{k=1}^{n-1}F_{k+2}F_{k-1}=F_n^2-1$$

$$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = A^{n-k} \begin{bmatrix} F_k \\ F_{k-1} \end{bmatrix} = \begin{bmatrix} F_{n-k+1} & F_{n-k} \\ F_{n-k} & F_{n-k-1} \end{bmatrix} \begin{bmatrix} F_k \\ F_{k-1} \end{bmatrix}$$
$$F_n = F_k F_{n-k+1} + F_{n-k} F_{k-1}$$

换元即得

$$F_{a+b} = F_a F_{b+1} + F_b F_{a-1} = F_a F_{b-1} + F_b F_{a+1}$$

考虑$|A^n| = |A|^n$，于是有

$$|A^n| = |A|^n = F_{n+1}F_{n-1} - F_n^2 = (-1)^n |A^n| = |A|^n = F_{n+1}F_{n-1} - F_n^2 = (-1)^n$$

性质：$\sum_{k=1}^{n} F_k F_{k+1}$

（见线性递推-特征值分解部分）

A的特征多项式为$p(\lambda) = (1-\lambda)(-\lambda) - 1$。令$p(\lambda) = 0$解得$\lambda = \frac{\pm\sqrt{5}+1}{2}$。

设其两个特征值为λ_1, λ_2，则有$\lambda_1 \lambda_2 = -1, \lambda_1 + \lambda_2 = 1$

$$P = \begin{bmatrix} \lambda_1 & \lambda_2 \\ 1 & 1 \end{bmatrix}; P^{-1} = \frac{1}{\sqrt{5}} \begin{bmatrix} -1 & \lambda_2 \\ 1 & -\lambda_1 \end{bmatrix}$$

$$A^k = \frac{1}{\sqrt{5}} \begin{bmatrix} \lambda_1 & \lambda_2 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \lambda_1^k & 0 \\ 0 & \lambda_2^k \end{bmatrix} \begin{bmatrix} -1 & \lambda_2 \\ 1 & -\lambda_1 \end{bmatrix} = \frac{1}{\sqrt{5}} \begin{bmatrix} \lambda_1^{k+1} & \lambda_2^{k+1} \\ \lambda_1^k & \lambda_2^k \end{bmatrix} \begin{bmatrix} -1 & \lambda_2 \\ 1 & -\lambda_1 \end{bmatrix}$$
$$= \frac{1}{\sqrt{5}} \begin{bmatrix} \lambda_2^{k+1} - \lambda_1^{k+1} & \lambda_1^{k+1}\lambda_2 - \lambda_2^{k+1}\lambda_1 \\ \lambda_2^k - \lambda_1^k & \lambda_1^k \lambda_2 - \lambda_2^k \lambda_1 \end{bmatrix} = \frac{1}{\sqrt{5}} \begin{bmatrix} \lambda_2^{k+1} - \lambda_1^{k+1} & \lambda_2^k - \lambda_1^k \\ \lambda_2^k - \lambda_1^k & \lambda_2^{k-1} - \lambda_1^{k-1} \end{bmatrix}$$

$$F_n = \frac{1}{5}(\lambda_1^n - \lambda_2^n) = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right]$$
$$F_n^2 = \frac{1}{5}\left(\lambda_1^{2n} + \lambda_2^{2n} - 2(-1)^n\right)$$

$$\sum_{k=1}^{n} F_{2k-1} = F_{2n}$$
$$\sum_{k=1}^{n} F_{2k} = F_{2n+1} - 1$$
$$F_{n-1}F_{n+1} = F_n^2 + (-1)^n$$
$$\gcd(F_n, F_m) = F_{\gcd(n,m)}$$
$$n|m \Leftrightarrow F(n)|F(m)$$

$Fib_0 = 0, Fib_1 = 1$时的OGF

$$F(x) = xF(x) + x^2 F(x) + x$$
$$F(x) = \frac{x}{1 - x - x^2}$$

Catalan数

$n=$	0	1	2	3	4	5	6	7	8	9	10
C_n	1	1	2	5	14	42	132	429	1430	4862	16796

性质：

$$C_n=\frac{1}{n+1}\binom{2n}{n}=\prod_{k=2}^{n}\left(1+\frac{n}{k}\right)=\sum_{k=0}^{n-1}C_kC_{n-k-1}$$

OGF：

$$F(x)=1+xF(x)^2=\frac{1-\sqrt{1-4x}}{2x}=\frac{2}{1+\sqrt{1-4x}}$$

Motzkin数

n=	0	1	2	3	4	5	6	7	8	9	10
M_n	1	1	2	4	9	21	51	127	323	835	2188

定义：M_n表示从$(0,0)$开始只能走$(1,1),(1,-1),(1,0)$且不走到第四象限的情况下走到$(n,0)$的方案数。

M_n表示在n个点的圆上画出数条不相交弦的全部方法的总数。

性质：

$$M_n=\frac{(2(n-1)+3)M_{n-1}+(3(n-2)+3)M_{n-2}}{n+2}$$

$$M_n=M_{n-1}+\sum_{k=0}^{n-2}M_kM_{n-2-k}=\sum_{k=0}^{\left\lfloor\frac{n}{2}\right\rfloor}\binom{n}{2k}C_k$$

OGF:

$$F(x)=xF(x)+x^2F^2(x)=\frac{1-x-\sqrt{1-2x-3x^2}}{2x^2}=\frac{2}{1-x+\sqrt{1-2x-3x^2}}$$

注：C_k为卡塔兰数。

Delannoy数

定义：D_n表示从$(0,0)$开始只能走$(0,1),(1,0),(1,1)$走到(n,n)的方案数。

枚举走了i个$(1,1)$可得

$$D_n=\sum_{i=0}^{n}\binom{2n-i}{i}\binom{2n-2i}{n-i}=\sum_{i=0}^{n}\binom{n+i}{n-i}\binom{2i}{i}$$

$$D_n={}_2F_1(-n,n+1,1,-1)$$

OGF:

$$D(x)=\frac{1}{\sqrt{1-6x+x^2}}$$

第一类Stirling数

n/m	0	1	2	3	4	5	6	7	8
0	1								
1	0	1							
2	0	1	1						
3	0	2	3	1					
4	0	6	11	6	1				
5	0	24	50	35	10	1			
6	0	120	274	225	85	15	1		
7	0	720	1764	1624	735	175	21	1	
8	0	5040	13068	13132	6769	1960	322	28	1

定义：$s(n,m)$表示将大小为n的集合划分成m个圆排列的方案数

考虑n所在排列（加入某个圆排列或单独作为一个新的圆排列）可得到递推式：

$$s(n,m)=s(n-1,m-1)+(n-1)s(n-1,m)$$

特别的，$s(n,0)=[n==0]$

注：每个圆排列可被视作一个置换，因而

$$\sum_{k=0}^{n}s(n,k)=n!$$

行的OGF：

由递推式有

$$s_n(x)=xs_{n-1}(x)+(n-1)s_{n-1}=(x+n-1)s_{n-1}(x)=\prod_{i=0}^{n-1}(x+i)$$

列的EGF:

单一圆排列的EGF为

$$\sum_{k=1}^{\infty}\frac{(k-1)!x^k}{k!}=\sum_{k=1}^{\infty}\frac{x^k}{k}=-\ln(1-x)$$

卷m次后除去$m!$消除圆排列之间的先后顺序即可得到第一类斯特林数的EGF

$$s_m(x)=\frac{(-\ln(1-x))^m}{m!}$$

注：上式是无符号第一类斯特林数的EGF，带符号的为

$$s_m(x)=\frac{(\ln(1+x))^m}{m!}$$

第二类Stirling数

n/m	0	1	2	3	4	5	6	7	8
0	1								
1	0	1							
2	0	1	1						
3	0	1	3	1					
4	0	1	7	6	1				
5	0	1	15	25	10	1			
6	0	1	31	90	65	15	1		
7	0	1	63	301	350	140	21	1	
8	0	1	127	966	1701	1050	266	28	1

定义：$S(n,m)$表示将大小为n的集合划分成m个非空集合的方案数

考虑n所在集合（加入m个集合中的一个或单独作为一个新的集合）可得到递推式：

$$S(n,m)=S(n-1,m-1)+mS(n-1,m)$$

考虑容斥，给每个集合编号后设A_i为第i个集合为空的放法，则所求为$\bar{A}_i$的交，即

$$m!S(n,m)=m^n-\sum_{k=1}^{m}(-1)^{k-1}\binom{m}{k}(m-k)^n$$

得

$$S(n,m)=\frac{1}{m!}\sum_{k=0}^{m}(-1)^{k-1}\binom{m}{k}(m-k)^n$$

性质：

考虑将m个物品分入n个盒子的方案数，枚举有k个有物品的盒子，可得

$$m^n=\sum_{k=0}^{m}S(m,k)\binom{n}{k}k!$$

二项式反演可得

$$k!S(m,k)=\sum_{n=0}^{k}(-1)^{k-n}\binom{k}{n}n^m$$

此式与上式等价。

行的OGF:

由通项公式得

$$S_n(x)=\sum_{k=0}^{n}\frac{x^k}{k!}\sum_{i=0}^{k}(-1)^i\binom{k}{i}(k-i)^n=\sum_{k=0}^{n}x^k\left(\sum_{i=0}^{k}\frac{(-1)^i}{i!}\frac{(k-i)^n}{(k-i)!}\right)$$

列的OGF:由递推式有：

$$S_m(x)=xS_{m-1}(x)+mxS_m(x)$$

得

$$S_m(x)=\frac{x}{1-mx}S_{m-1}(x)=\frac{x^m}{\prod_{i=1}^m 1-ix}$$

Bell数

n=	0	1	2	3	4	5	6	7	8	9	10
$Bell_n$	1	1	2	5	15	52	203	877	4140	21147	115975

定义：贝尔数B_n表示将n化为至少一个非空集合的方案数。

性质：

$Bell_n=\sum_{k=1}^n S(n,k)$

注：$S(n,k)$为第二类斯特林数。

EGF: $Bell(x)=e^{e^x-1}$

Bernoulli与等幂求和

n=	0	1	2	3	4	5	6	7	8	9	10
B_n	1	$-\frac{1}{2}$	$\frac{1}{6}$	0	$-\frac{1}{30}$	0	$\frac{1}{42}$	0	$-\frac{1}{30}$	0	$\frac{5}{66}$

定义（伯努利数）：

$$B_0=1,\sum_{i=0}^n\binom{n+1}{i}B_i=0\quad(n\geq 1)$$

由其递归定义推导EGF$B(x)$：

$$\sum_{i=0}^{n-1}\binom{n}{i}B_i+B_n=B_n\quad(n\geq 2)$$

$$\sum_{i=0}^{n}\binom{n}{i}B_i=B_n\quad(n\geq 2)$$

当$n=1$时有

$$\binom{1}{0}B_0+\binom{1}{1}B_1=\frac{1}{2}=-\frac{1}{2}+1$$

因此有

$$e^xB(x)=B(x)+x$$

$$B(x)=\frac{x}{e^x-1}$$

定义（等幂求和）：

$$S_k(n)=\sum_{i=0}^{n-1}i^k$$

设$F_n(x)[\frac{x^k}{k!}]=S_k(n)$

$$F_n(x)=\sum_{i=0}^{\infty}S_k(n)\frac{x^k}{k!}=\sum_{k=0}^{\infty}\sum_{i=0}^{n-1}i^k\frac{x^k}{k!}=\sum_{i=0}^{n-1}\sum_{k=0}^{\infty}\frac{(xi)^k}{k!}=\sum_{i=0}^{n-1}e^{xi}=\frac{e^{xn}-1}{e^x-1}$$

观察到

$$F_n(x)=B(x)\frac{e^{xn}-1}{x}$$

因为

$$\frac{e^{xn}-1}{x}=\frac{1}{x}\left(-1+\sum_{i=0}^{\infty}\frac{(nx)^i}{i!}\right)=\frac{1}{x}\sum_{i=1}^{\infty}\frac{n^ix^i}{i!}=\sum_{i=0}^{\infty}\frac{n^{i+1}}{i+1}\frac{x^i}{i!}$$

所以

$$S_k(n)=F_n(x)[x^k]=\sum_{i=0}^{k}\binom{k}{i}\frac{n^{k-i+1}}{k-i+1}B_i=\frac{1}{k+1}\sum_{i=0}^{k}\binom{k+1}{i}B_in^{k-i+1}$$

是一个关于n的$k+1$次多项式。

Narayana数

n\k	1	2	3	4	5	6	7	8
1	1							
2	1	1						
3	1	3	1					
4	1	6	6	1				
5	1	10	20	10	1			
6	1	15	50	50	15	1		
7	1	21	105	175	105	21	1	
8	1	28	196	490	490	196	28	1

定义：$N(n,k)$表示长度为$2n$的合法括号序列中有k对直接相邻的左右括号的方案数。

性质：

$$N(n,k)=\frac{1}{n}\binom{n}{k-1}\binom{n}{k}$$

$$\sum_{k=1}^{n}N(n,k)=C_n$$

注：C_k为卡塔兰数。

Euler数

拆(划)分数与五边形数

$n=$	1	2	3	4	5	6	7	8	9	10	11
$p(n)$	1	2	3	5	7	11	15	22	30	42	56

$n=$	12	15	20	30	40	50	60	70	80	90	100
$p(n)$	77	176	627	5604	37338	204226	966467	4087968	15796476	56634173	190569292

拆分数：p_n为将n拆成数个正整数的方案。$p_{n,k}$为将n拆成恰好k个正整数的方案。

不难列出拆分数的OGF:

$$F(x)=(1+x+x^2+\cdots)(1+x^2+x^4+\cdots)+\ldots=\prod_{i=1}^{\infty}\sum_{j=0}^{\infty}x^{ij}=\prod_{i=1}^{\infty}\frac{1}{1-x^i}$$

和五边形数相关的神奇函数：

$$P(x)=\prod_{i=1}^{\infty}(1-x^i)=\sum_{i=0}^{\infty}(-1)^i x^{\frac{1}{2}i(3i\pm1)}$$

易得

$$F(x)P(x)=1$$

若限定每个数至多用k次，则有OGF:

$$F_k(x)=\prod_{i=1}^{\infty}\sum_{j=0}^{k}x^{ij}=\prod_{i=1}^{\infty}\frac{1-x^{(k+1)i}}{1-x^i}=P(x^{k+1})F(x)$$

注：$P(x)$前n项的系数只有$O(\sqrt{n})$个非零项，因此在预处理出$F(x)$的条件下可在$O(\sqrt{n})$内算出$F_k(x)$的第n项系数。

更一般的拆分数：$p(n,k,W,R)$等于将n分成k个正整数的和且每个正整数都属于R且出现次数属于W。

OGF:

$$\sum_{n=1}^{\infty}\sum_{k=1}^{\infty}p(n,k,W,R)x^n y^k=\prod_{r\in R}\sum_{w\in W}y^w x^{rw}$$

命题：将n拆成不超过k个正整数的方案数等于将n拆成不大于k的正整数的方案数。

证明：

将n拆成不超过k个正整数的方案数有生成函数

$$[x^n y^k]\prod_{r=0}^{\infty}\sum_{w=0}^{\infty}y^w x^{rw}=[x^n y^k]\prod_{r=0}^{\infty}\frac{1}{1-yx^r}$$

将n拆成不大于k的正整数的方案数有生成函数

$$[x^n]\prod_{r=1}^{k}\sum_{w=0}^{\infty}x^{rw}=[x^n]\prod_{r=1}^{k}\frac{1}{1-x^r}$$

因此有

$$F(x,y)=\prod_{r=0}^{\infty}\sum_{w=0}^{\infty}y^w x^{rw}=\prod_{r=0}^{\infty}\frac{1}{1-yx^r}$$

$$G(x,y)=\sum_{i=0}^{\infty}y^i\prod_{r=1}^{i}\sum_{w=0}^{\infty}x^{rw}=\sum_{i=0}^{\infty}\prod_{r=1}^{i}\sum_{w=0}^{\infty}yx^{rw}=\sum_{i=0}^{\infty}\prod_{r=1}^{i}\frac{y}{1-x^r}$$

常见计数模型

球盒(十二模式)

将n个球放到m个盒子里，球可/不可区分，盒子可/不可区分，映射无限制/单射/满射。

小球不可区分，盒子可区分，映射是单射：

等价于在m个盒子中选择n个放球，答案为$\binom{m}{n}$

小球不可区分，盒子可区分，映射是满射：

等价于在长度为n的序列中的$n-1$个空隙中插入$m-1$个隔板，因此答案为$\binom{n-1}{m-1}$

小球不可区分，盒子可区分，映射无限制：

等价于在长度为$n+m$的序列中的$n+m-1$个空隙中插入$m-1$个隔板，因此答案为$\binom{n+m-1}{m-1}$

生成函数为

$$[x^n](1+x+x^2+\cdots+x^n)^m=[x^n]\left(\frac{x^{n+1}-1}{x-1}\right)^m$$

小球可区分，盒子可区分，映射是单射：

等价于从m个盒子中选择n个进行排列，答案为$m(m-1)\cdots(m-n+1)=m!/(m-n)!$

小球可区分，盒子可区分，映射是满射：

等价于将n个元素划分为m个非空集合后对每个集合进行标号，答案为$\left\{ {n \atop m} \right\}m!$

小球可区分，盒子可区分，映射无限制：

每个球有m种策略，互相独立，因此答案为m^n种。

小球不可区分，盒子不可区分，映射是单射：

若$n>m$，则答案为0，否则答案为1。

小球不可区分，盒子不可区分，映射是满射：

等价于将整数n拆分成恰好m个正整数的方案数，即$p_{n,m}$。

小球不可区分，盒子不可区分，映射无限制：

等价于将整数n分拆成至多m个正整数的方案数。等价于将$n+m$拆分成恰好m个正整数的方案数，即$p_{n+m,m}$。

小球可区分，盒子不可区分，映射是单射：

若$n > m$，则答案为0，否则答案为1。

小球可区分，盒子不可区分，映射是满射：

等价于将n个元素划分为m个非空集合的方案数，答案为$\begin{Bmatrix} n \\ m \end{Bmatrix}$。

小球可区分，盒子不可区分，映射无限制：

等价于将n个元素划分为至多m个非空集合的方案数，答案为$\sum_{k=1}^{m} \begin{Bmatrix} n \\ k \end{Bmatrix}$。

整数分拆

定义：

拆分数p_n是将n拆分为数个无序正整数之和的方案数。

拆分数$p_{n,k}$是将n拆分成k个无序正整数之和的方案数。

命题：将n拆分成k个无序正整数之和的方案数等于将$n-k$拆分成至多k个正整数之和的方案数。

证明：可以构造一个双射。对于前者的每个方案，将每个数减去1。对于后者的每个方案，将每个数加上1并用1补齐至k个。

定义(Ferrers图)：对于任意一个拆分方案，设其中第i大的数是x_i，则其对应的Ferrers图的第i行有x^i个点。

如$12=5+3+3+1$，则其Ferrers图为

```
00000
000
000
0
```

命题：将n拆分成至多k个正整数之和的方案数等于将n拆分成数个不超过k的正整数之和的方案数。

证明：前者的每个方案的Ferrers图至多有k行，后者的每个方案的Ferrers图有至多k列。这构成了一个双射，映射即为Ferrers图的转置。

考虑其生成函数，枚举i被用了j次：

$$F(x)=\sum_{i=1}^{\infty} p_i x^i=\prod_{i=1}^{\infty}\sum_{j=0}^{\infty} x^{ij}=\prod_{i=1}^{\infty}\frac{1}{1-x^i}$$

$$F_k(x)=\sum_{i=1}^{\infty} p_{i+k,k} x^i=\prod_{i=1}^{k}\sum_{j=0}^{\infty} x^{ij}=\prod_{i=1}^{k}\frac{1}{1-x^i}$$

命题：将n拆分成数个奇数之和的方案数等于将n拆分成数个互不相同的数之和的方案数。

证明1：构造一个双射。

对于前者的每个方案，若方案中某个奇数w出现了x次，考虑x的二进制展开$x=b_0 2^0+b_1 2^1+\cdots$，若b_i为1则往方案中加一个$2^i w$。这是一个单射。

对于后者的每个方案，若方案中存在某个偶数$2^k w$，则将其拆成2^k个w。这也是一个单射。证毕。

证明2：考虑生成函数。

前者的生成函数:

$$F(x)=\prod_{i=1}^{\infty}\frac{1}{1-x^{2i-1}}$$

后者的生成函数：

$$G(x)=\prod_{i=1}^{\infty}(1+x^i)=\prod_{i=1}^{\infty}\frac{1-x^{2i}}{1-x^i}=\prod_{i=1}^{\infty}(1-x^{2i})\Big/\prod_{i=1}^{\infty}(1-x^i)=\prod_{i=1}^{\infty}\frac{1}{1-x^{2i-1}}=F(x)$$

命题：将n拆分成数个模3不为0的正整数的方案数等于将n拆分成每种数出现至多两次的方案数。

证明：考虑生成函数。

前者的生成函数：

$$F(x)=\prod_{i=1}^{\infty}\frac{1}{1-x^{3i-1}}\frac{1}{1-x^{3i-2}}$$

后者的生成函数：

$$G(x)=\prod_{i=1}^{\infty}(1+x^i+x^{2i})=\prod_{i=1}^{\infty}\frac{1-x^{3i}}{1-x^i}=F(x)$$

格路径

考虑一个有n行m列的网格。从左下角$(0,0)$开始，每次只能往右/上方走。

无限制

从$(0,0)$走到(n,m)共有$\binom{n+m}{n}$种走法。

证明：一共只走$n+m$步，其中恰好n步向右走，恰好m步向上走。

因此每种走法唯一对应一个有n个0和m个1的01序列，其中0代表向上走，1代表向右走。

这样的方案数为$\binom{n+m}{n}$。

有一个限制

从$(0,0)$走到(n,m)且不经过直线$l:y=x+k$上的任何一个点的方案数为$\binom{n+m}{n}-\binom{n+m}{n+k}$，其中$m<n+k$且$k>0$，或$m>n+k$且$k<0$。

证明：对于任意一个不合法方案，设其第一次经过l_1的位置为点p，将该方案对应的折线p以后的部分沿l_1翻转，可以得到一个从$(0,0)$走到$(m-k,n+k)$的方案。不难证明这是一个双射。

有两个限制

计算从$(0,0)$走到(n,m)且不经过直线$l_1:y=x+k_1$和$l_2:y=x+k_2$上的任何一个点的方案数，其中$n+k_1<m<n+k_2$且$k_1<0<k_2$。

考虑容斥。

对于每个方案，按其经过l_1和l_2的顺序可以对应一个交错12串。即每连续数次经过l_1则加上一个1，每连续数次经过l_2则加上一个2。

如先经过两次l_1，再经过三次l_2，再经过一次l_1的方案对应的串为121。

考虑推广前面仅有一个限制时的做法。

将(n,m)翻转到其沿l_1的对称点，对应的是在终点前连续数次经过l_1的方案，即以1或12结尾的所有方案。

在这个基础上再沿l_2翻转一次，对应的是在终点前连续数次经过l_2再连续数次经过l_1的方案，对应的是以21和以212结尾的所有方案。

将(n,m)翻转到其沿l_2的对称点，对应的是在终点前经过了一次l_2的方案，即以2或21结尾的所有方案。

在这个基础上再沿l_1翻转一次，对应的是在终点前连续数次经过l_1再连续数次经过l_2的方案，对应的是以12和以121结尾的所有方案。

依此类推，翻转奇数次的所有方案减去翻转偶数次的所有方案即是以1结尾或以2结尾的所有方案之和，即不合法方案数。

```
int cal(int n, int m, int k1, int k2) {
    if (m <= n + k1 || m >= n + k2 || k1 >= 0 || k2 <= 0) return 0;
    int res = bi(n + m, n), d[2] = { k1, -k2 };
    for (int x = n + d[0], t = 0; x >= 0 && x <= n + m; x += d[t ^= 1])
        res = (t ? add : sub)(res, bi(n + m, x));
    for (int x = m + d[1], t = 1; x >= 0 && x <= n + m; x += d[t ^= 1])
        res = (t ? sub : add)(res, bi(n + m, x));
    return res;
}
```

一些计数定理

Burnside引理

给定集合S，其在群G作用下的轨道数为

$$\frac{1}{|G|}\sum_{g\in G}|X^g|$$

其中$X_g=\{x\in S|gx=x\}$，即在元素g作用下的不动点集合。

x是置换g的不动点当且仅当g的每个轮换中的所有元素均相同。

例：n个点的环，每个点可以染c种颜色，问不同染色方案数。方案不同当且仅当旋转任意角度后两个环上存在至少一个颜色不相同的同位置的点。

置换群为$\mathbb{Z}_n$，设其生成元为g，则g^d由$\gcd(n,d)$个长度为$n/\gcd(n,d)$的轮换组成。

与其相同的置换有$\varphi(n/d)$种，因此方案数为

$$\frac{1}{n}\sum_{d|n}\varphi(n/d)c^d$$

LGV引理

矩阵树定理

无向图

以下默认图中无自环。

定义（拉普拉斯矩阵）：无向图$G=(V,E)$的拉普拉斯矩阵被定义为$L(G)=D-A$，其中

D为度数矩阵，即对角元D_{ii}为$\deg(i)$的矩阵。

A为邻接矩阵，即A_{ij} 为i,j之间的边数。

定义：

$T(G)$为G的生成树数量。

$L(G)[u]$为$L(G)$删去u对应行列后所得矩阵。类似的，$L(G)[u,v]$为$L(G)$分别删去u,v对应行列后所得矩阵。

命题：$|V|-\operatorname{rank}L(G)$为$G$的连通块数量。

证明略。

定理(Kirchhoff)：$\forall u\in V,T(G)=\det L(G)[u]$。即无向图$G$的生成树数量为其拉普拉斯矩阵划去任意节点对应行列所得矩阵的行列式之值。

证明：

当G不连通时：按连通块可将$L(G)$划分成许多块。划去任意节点对应的行列后，余下来的每块行列和均为0，因此$\det L(G)[u]=0$。

当G连通时，对图中的边数做归纳：

当$|E|=0$时显然，否则存在一条边$(u,v)\in E$。

该图中的生成树可以根据是否包含(u,v)分为两类。

不包含(u,v)的生成树数量等于$T(G\setminus(u,v))$。其中$G\setminus(u,v)$为将(u,v)删去后的G。

包含(u,v)的生成树数量等于$T(G/(u,v))$。其中$G/(u,v)$为将u,v两点粘合成一点的G。

对$L(G)[u]$中点v对应行展开，设C_{vw}为$(L(G)[u])_{vw}$对应的代数余子式，可得

$$\begin{aligned}\det L(G)[u]&=\sum_w(L(G)[u])_{vw}C_{vw}\\&=(L(G)[u])_{vv}C_{vv}+\sum_{w\neq v}(L(G)[u])_{vw}C_{vw}\end{aligned}$$

注意到$L(G\setminus(u,v))[u]$与$L(G)[u]$的区别仅在于(v,v)位置上的元素多了1，于是有

$$\begin{aligned}(L(G)[u])_{vv}&=(L(G\setminus(u,v))[u])_{vv}+1\\\forall w\neq v,(L(G)[u])_{vw}&=(L(G\setminus(u,v))[u])_{vw}\end{aligned}$$

代入前式有

$$\begin{aligned}&=((L(G\setminus(u,v))[u])_{vv}+1)C_{vv}+\sum_{w\neq v}(L(G\setminus(u,v))[u])_{vw}C_{vw}\\&=C_{vv}+\sum_w(L(G\setminus(u,v))[u])_{vw}C_{vw}\end{aligned}$$

注意到

1. C_{vv}为$L(G)$删去u和v对应行列所得行列式，其恰好等于将u,v两点粘合成一点(u,v)后所得图$G/(u,v)$的拉普拉斯矩阵删去(u,v)对应行列所得矩阵$L(G/(u,v))[(u,v)]$，因此$\det C_{vv}$即为G中包含边(u,v)的生成树数量。

2. 而C_{vw}等于$(L(G\setminus(u,v)))_{vw}$对应的代数余子式，因此后半部分即为$\det L(G\setminus(u,v))[u]$，即为$G$中不包含$(u,v)$的生成树数量。

$$T(G)=\det L(G)[u]=\det L(G/(u,v))[(u,v)]+\det L(G\setminus(u,v))[u]=T(G\setminus(u,v))+T(G/(u,v))$$

注：本证明译自某谱图论专著相关讲义

若将两点之间的边数视为权值并定义生成树的权值为边权之积，则矩阵树统计的是所有生成树的权值和。

例（洛谷P5296 [北京省选集训2019]生成树计数）：定义生成树权值为边权之和，计算所有生成树的权值的k次方之和。

可以通过将边权转化为指数函数的低阶展开来将乘法转化为加法。即将权为w的边变成权为$\exp wx$的边。不难发现

$$(\exp w_1x)(\exp w_2x)=\exp((w_1+w_2)x)$$

若求的是权值和的k次方之和，则卷积时保留前k次方即可，答案即为结果的k次项乘上$k!$。

有向图

类比无向图，可以得出有向图的矩阵树定理。

有向图中的矩阵树定理计算的是两类树形图的数量。

定义：

根向树形图：根的出度为0，其他所有点出度均为1的无环子图。

叶向树形图：根的入度为0，其他所有点入度均为1的无环子图。

矩阵树定理（有向图形式）：

定义：设

D^{out}为出度矩阵，即对角元D^{out}_{ii}为点i的出度（从点i指向其他点的边数）的矩阵。

D^{in}为入度矩阵，即对角元$D^{in}_i i$为点i的入度（从其他点指向点i的边数）的矩阵。

$$L^{out}(G)=D^{out}(G)-A(G)$$
$$L^{in}(G)=D^{in}(G)-A(G)$$

$T^{out}_r(G)$为有向图G中以r为根的根向树形图数量

$T^{in}_r(G)$为有向图G中以r为根的叶向树形图数量。

则有

$$T^{out}_r(G)=\det L^{out}(G)[r]$$
$$T^{in}_r(G)=\det L^{in}(G)[r]$$

证明：先证根向树形图相关的。对点数做归纳，两个点时显然成立。

否则对于某个点u，该图中的根向树形图分为两类。

包含边(u,r)的根向树形图数量等于$a_{ur}T(G/(u,r))$，其中$G/(u,r)$是将图中指向u的所有边改为指向r并将点u删去所得的图。

不包含边(u,r)的根向树形图数量等于$T(G\setminus(u,r))$，其中$G\setminus(u,r)$是将边(u,r)删去后所得的图。

进行类似的观察，可以发现$L^{out}(G)[r]$和$L^{out}(G/(u,r))[r]$之间的区别仅在于删去了点u所在行列，而$L^{out}(G)[r]$和$L^{out}(G\setminus(u,r))[r]$之间的区别仅在于对角线上$u$位置处的元素多了$a_{ur}$。

对u所在行展开即得

$$\begin{aligned} &T_r^{out}(G) = \det L^{out}(G)[r] \\ &= \det L^{out}(G/(u,r))[r] + \det L^{out}(G \setminus (u,r))[r] \\ &= T_r^{out}(G/(u,r)) + T_r^{out}(G \setminus (u,r)) \end{aligned}$$

对于叶向树形图，进行类似的分析后对任意一个r以外的点u所在列展开即得证。

BEST定理

定理(BEST)：有向欧拉图$G = (V, E)$的欧拉回路个数$ec(G)$满足

$$ec(G) = T_r^{out} \prod_{v \in V} (\delta^{out}(v) - 1)!$$

其中r是G中的任意一个点。

证明：可以构造一个从一个树形图到一组数量为$\prod_{v \in V} (\delta^{out}(v) - 1)!$的欧拉回路的双射。

先钦定一个根r。对于任意一个欧拉回路，钦定其从r出发经过的第一条边为(r, r')。对于V中的任意一个点，其在该欧拉回路的边列表中的最后一次出边组成的图是一个树形图。固定这些出边后，其他所有边的顺序可以任意选择。这样的方案数即$\prod_{v \in V} (\delta^{out}(v) - 1)!$。

图计数

Prufer序列

定义：一棵有标号树的Prufer序列为递归地将其最小的叶节点删去并将其邻点标号加入序列末尾直到剩余两个点所得序列。

Prufer序列转化为树：

维护一个集合S，初始$S = \{1, 2, \ldots, n\}$。对于序列中从左到右的每个元素u，找到未在其右侧出现的S中的最小元素v并将u, v连边，然后将v从S中删去，最后将S中剩余两个点连边即可完成对树的还原。

扩展——森林的Prufer序列：

定义：一个n个点的有k个连通块组成的森林的Prufer序列与树的构造方式几乎相同，唯一的区别是剩余$k+1$个点时停止。易得n个点k个连通块组成的森林的$Prufer$序列长度为$n-k-1$

1. 先钦定k个根
2. 第$n-k$个位置的值为k个根之一。
3. 前$n-k-1$个位置上的值任意。

不难得到n个点，有k个连通块的森林数量为$\binom{n}{k} k n^{n-k-1}$

有标号无根树计数

定理(Caylay)：n个点的有标号无根树共有n^{n-2}种。从Prufer序列的构造与还原不难看出一棵树对应着一个Prufer序列，而每个Prufer序列在其$n-2$个位置均有n种取值可能。

法2：

枚举叶子个数k，可得到

当$0 \le n \le 2$时有$F(n) = 1$，否则有$F(n) = \sum_{k=1}^{n-1} \binom{n}{k} (n-k)^k F(n-k)$

有标号有根树计数

给每个有标号无根树定一个根即可得到有标号有根树，因而n个点的有标号有根树共有n^{n-1}种。

有标号基环树计数

法1：考虑魔改Prufer序列，先钦定k个点在环上，方案为$\binom{n}{k}$。

将环上的边都断开并将所有环上的点连向一个标号为正无穷的特殊点。

该树的Prufer序列具有如下性质：

1. 最后k个位置的值均为正无穷。
2. 第$n-k$个位置的值必为环上的k个点之一。
3. 前$n-k-1$个位置上的值任意。

0k个点的环排列有$\frac{1}{2}k!$种，因而n个点的基环树个数为

$$\sum_{k=3}^{n}\binom{n}{k}\frac{k!}{2}n^{n-k-1}=\frac{1}{2}\sum_{k=3}^{n}\frac{n!}{(n-k)!}n^{n-k-1}$$

考虑n个点基环树数量的OGF:

$$F(x)=\frac{1}{2}\sum_{i=0}^{\infty}\frac{x^i}{i!}\sum_{k=3}^{i}\frac{i!}{(i-k)!}i^{i-k-1}=\frac{1}{2}\sum_{i=0}^{\infty}\frac{x^i}{i!}\sum_{k=0}^{i-3}\frac{i!}{k!}i^{k-1}=\frac{1}{2}\sum_{i=0}^{\infty}x^i\sum_{k=0}^{i-3}\frac{i^{k-1}}{k!}$$

法2：

考虑有标号的每个点有恰好1条出边，可以存在自环的有向图，其每个连通块去掉方向后可以分为三类：

1. 基环树
2. 有自环的树
3. 有重边的树

因此考虑这类有向图计数的EGF：

$$F(x)=1+x+2^2x^2+3^3x^3=\sum_{i=0}^{\infty}i^ix^i$$

对其取log可得上面三类图数量之和对应的生成函数。

因为基环树的环有两种定向方法，有自环的树有恰好$n\times n^{n-2}$种，有重边的树有恰好$(n-1)\times n^{n-2}$种，所以n个点的基环树数量为

$$\frac{1}{2}\left([x^n]\log F-(2n-1)n^{n-2}\right)$$

无标号有根树计数

钦定根后剩下的子树方案数相当于一个完全背包。

考虑欧拉变换的定义可写出OGF相关的方程：

$$f(x)=x\mathscr{E}(f(x))=x\exp\left(\sum_{j=1}^{\infty}\frac{f(x^j)}{j}\right)$$

两边取对数

$$\ln f(x)=\ln x+\sum_{j=1}^{\infty}\frac{f(x^j)}{j}$$

两边求导

$$\frac{f'(x)}{f(x)}=\frac{1}{x}+\sum_{j=1}^{\infty}x^{j-1}f'(x^j)$$

$$xf'(x)=f(x)\left(1+\sum_{j=1}^{\infty}x^jf'(x^j)\right)$$

设

$$g(x)=\sum_{j=1}^{\infty}x^jf'(x^j)$$

可得

$$b_i=\sum_{j|i}ia_i,ia_i=a_i+\sum_{j=1}^{i-1}a_jb_{i-j},a_i=\frac{1}{i-1}\sum_{j=1}^{i-1}a_jb_{i-j}$$

注：这里$a_0=b_0=0,a_1=1$。

使用有依赖的分治卷积，计算`f`和`g`的部分为

```
g[l] = (l == 1 ? 1 : mul(g[l], inv(l - 1)));
for (int j = l; j < f.size(); j += l)
    f[j] = add(f[j], mul(l, g[l]));
```

然后直接`conv(f,g,1,n+1)`后`g`即为所求。

烷基计数

即度数限制为3的有根树。

因为无标号，所以使用Burnside引理去重。

置换群为S_3，e对应的方案数为$F(x)^3$，(123)和(132)对应的方案数为$2F(x^3)$，$(12),(23),(31)$对应的方案数分别为$3F(x)F(x^2)$。

因此有

$$F(x)=1+x\frac{F(x)^3+2F(x^3)+3F(x)F(x^2)}{6}$$

牛顿迭代时$F(x^2)$与$F(x^3)$的系数均已知，因此可以当作常数处理。构造方程：

$$t(F)=\frac{1}{6}x(F^3+2B+3FA)+1-F$$

则

$$t'(F)=\frac{1}{6}x(3F^2+3A)-1$$

得迭代式

$$G=F-\frac{x(F^3+2B+3FA)+6-6F}{x(3F^2+3A)-6}$$

```
vi alkyl(int n1) {
    if (n1 == 1) return vi(1, 1);
    int n2 = (n1 + 1) >> 1;
    vi f = alkyl(n2); f.resize(n1, 0);
    vi a = pmul(xpow(f, 2), 3), b = pmul(xpow(f, 3), 2);
    vi f2 = pmul(f, f, n1), f3 = pmul(f2, f, n1);
    vi h1 = padd(f3, padd(b, pmul(a, f, n1)));
    h1.insert(h1.begin(), 0);
    h1[0] = add(h1[0], 6);
    h1 = psub(h1, pmul(f, 6));
    vi h2 = padd(pmul(f2, 3), a);
    h2.insert(h2.begin(), 0);
    h2[0] = sub(h2[0], 6);
    return psub(f, pmul(h1, pinv(h2), n1));
}
```

例(JSOI2011 同分异构体计数)：

统计有多少个环上点数不超过m的n个点的烃。

环上每个点可以接上两个烷基。方案树即即根度数至多为2，其他点度数至多为3的有根树数量。生成函数为

$$G(x)=\frac{1}{2}x(F(x)^2+F(x^2))$$

接下来要把这种东西插到环上后去重。

对于固定的一个环长k，置换群为D_k。

对于形如g^n的元素，即Z_k中的元素，总方案数为

$$\sum_{d|k}\varphi(k/d)[x^{n/(k/d)}]G(x)^d$$

对于e以外的满足$g^2=e$的元素，若$k\equiv 1 \bmod 2$，则总方案数为

$$k\sum_{i=(k-1)/2}^{\lfloor(n-1)/2\rfloor}[x^i]G(x)^{(k-1)/2}[x^{n-2i}]G(x)$$

否则总方案数为

$$\frac{k}{2}\sum_{i=k/2-1}^{\lfloor(n-2)/2\rfloor}[x^i]G(x)^{k/2-1}[x^{n-2i}]G(x)$$

这一类对应对称轴过正k边形顶点的情况，左边为对称轴两边的方案数，右边为对称轴上两点的方案数。

若$n\equiv 2 \bmod 0$，则还需加上

$$\frac{k}{2}[x^{n/2}]G(x)^{k/2}$$

这一类对应对称轴不过正k边形顶点的情况。

预处理出$G(x)$的1至m次幂，加起来除以$2k$即为答案。

无标号无根树计数

统计方法为无标号有根树的数量减去根不是重心的无标号有根树的数量。

若根不是重心，则根有一个大小至少为$\left\lfloor \frac{n}{2} \right\rfloor + 1$的子树。枚举该子树大小可得

$$g_n = f_n - \sum_{i=\left\lfloor \frac{n}{2} \right\rfloor + 1}^{n-1} f_i f_{n-i}$$

注意到有两个重心的时候，有一类树被统计了两次。即有两个重心且两边不同构的树。该类树的数量为$\binom{f_{\frac{n}{2}}}{2}$，减去即可。

因此有

$$G(x) = F(x) - \frac{1}{2}\left(F(x^2) - (F(x) - 1)^2\right)$$

烷烃计数

先计算根度数至多为4其他点度数至多为3的有根树数量。

考虑Burnside引理，置换群为S_4。

e对应方案数为$F(x)^4$

$(12), (13), (14), (23), (24), (34)$对应方案数为$6F(x^2)F(x)^2$

$(12)(34), (13)(24), (14)(23)$对应方案数为$3F(x^2)^2$

$(123), (234), (134), (124), (132), (243), (143), (142)$对应方案数为$8F(x^3)F(x)$

$(1234), (1243), (1324), (1342), (1423), (1432)$对应方案数为$6F(x^4)$

得

$$G(x) = 1 + \frac{1}{24}x\left(F(x)^4 + 6F(x^2)F(x)^2 + 3F(x^2)^2 + 8F(x^3)F(x) + 6F(x^4)\right)$$

接下来考虑去重。

只有一个重心时，非重心点必有一个大于$\lfloor n/2 \rfloor$的子树。这一部分的方案数为

$$b_n - \sum_{i=\lfloor n/2 \rfloor + 1} a_i a_{n-i}$$

有两个重心时，有一类树被统计了两次，因此需减去$\binom{a_{n/2}}{2}$

最终的生成函数为

$$H(x) = G(x) + \frac{1}{2}(F(x^2) - (F(x) - 1)^2 - 1)$$

```
vi alkane(int n) {
    int n1 = n + 1;
    vi a = alkyl(n1);
    vi a2 = pmul(a, a, n1), a4 = pmul(a2, a2, n1);
    vi a_2 = xpow(a, 2);
    vi b = pmul(padd(a4, padd(pmul(pmul(a_2, a2, n1), 6),padd(pmul(pmul(a_2,
a_2, n1), 3),padd(pmul(pmul(xpow(a, 3), a, n1), 8), pmul(xpow(a, 4), 6)))))),
inv(24));
    b.insert(b.begin(), 1);
    b = padd(b, pmul(psub(psub(xpow(a, 2), {1}), pmul(psub(a, {1}), psub(a,
{1}), n1)), inv(2)));
    b.pop_back();
    return b;
}
```

有标号连通图计数

对于满足某些性质的图（如欧拉图，即顶点度数均为偶数），用可以不连通的所有方案推广到连通的所有方案的情况可通过将所有方案减去不连通的方案获得。如枚举1号点所在连通块大小k，则除1号点之外的点共有$\binom{n-1}{k-1}$种方案：

$$f_n = g_n - \sum_{k=1}^{n-1} \binom{n-1}{k-1} f_k g_{n-k}$$

也可通过指数生成函数进行计数，即$G(x) = \ln F(x)$，其中$F(x), G(x)$分别为f_n, g_n的EGF。具体见"$e^{F(x)}$的组合意义"

有标号二分图计数

法1：

设大小为n，有m个连通块的二分图数量为$F(n,m)$，则有：

$$F(n,m) = \sum_{k=1}^{n-m+1} \binom{n-1}{k-1} F(k,1) F(n-k, m-1)$$

设$G(n)$为大小为n，对顶点进行了黑白染色的二分图数量，钦定m个黑点余下为白点并任意连边可得:

$$G(n) = \sum_{m=0}^{n} \binom{n}{m} 2^{m(n-m)} = \sum_{m=1}^{n} 2^m F(n,m)$$

$$= 2^1 F(n,1) + \sum_{m=2}^{n} 2^m \sum_{k=1}^{n-m+1} \binom{n-1}{k-1} F(k,1) F(n-k, m-1)$$

$$= 2F(n,1) + \sum_{k=1}^{n-1} \binom{n-1}{k-1} F(k,1) \sum_{m=2}^{n-k+1} 2^m F(n-k, m-1)$$

$$= 2F(n,1) + 2\sum_{k=1}^{n-1} \binom{n-1}{k-1} F(k,1) \sum_{m=1}^{n-k} 2^m F(n-k, m)$$

$$= 2F(n,1) + 2\sum_{k=1}^{n-1} \binom{n-1}{k-1} F(k,1) G(n-k)$$

可得递推式:

$$F(n,1)=\frac{1}{2}G(n)-\sum_{k=1}^{n-1}\binom{n-1}{k-1}F(k,1)G(n-k)$$

注：

$$G(n)=\sum_{m=0}^{n}\binom{n}{m}2^{m(n-m)}=\sum_{m=0}^{n}\binom{n}{m}2^{nm-m^2}=\sum_{m=0}^{n}\frac{n!}{m!(n-m)!}2^{\frac{1}{2}(n^2+m^2-(n-m)^2)-m^2}$$

$$=n!2^{\frac{n^2}{2}}\sum_{m=0}^{n}\frac{1}{m!(n-m)!}2^{-\frac{(n-m)^2}{2}-\frac{m^2}{2}}=n!2^{\frac{n^2}{2}}\sum_{m=0}^{n}\frac{2^{-\frac{(n-m)^2}{2}}}{(n-m)!}\frac{2^{-\frac{m^2}{2}}}{m!}$$

法2：

考虑n个点的二染色的连通二分图的EGF为$H(x)$，二染色的二分图的EGF为$G(x)$，连通二分图的EGF为$F_1(x)$，二分图的EGF为$F_2(x)$，则显然$G(x)=\exp H(x)$，且$H(x)=2F_1(x)$,$F_2(x)=\exp F_1(x)$。

于是$F_2(x)=\exp F_1(x)=\exp\frac{1}{2}H(x)=\exp\frac{1}{2}\log G(x)=\sqrt{G(x)}$。

有标号DAG计数

注意到DAG中存在一类特殊的点，即出度为0的点，我们通过枚举这些点进行计算。

设$f_{n,S}$为n个点，S中每个点出度均为0，其他点出度均大于0的DAG数量。

设$g_{n,S}$为n个点，S中每个点出度均为0，其他点出度任意的DAG数量。

n个点的DAG数量即$g_{n,\emptyset}$。

由定义有

$$g_{n,S}=\sum_{T\supseteq S}f_{n,T}$$

则由容斥有

$$f_{n,S}=\sum_{T\supseteq S}(-1)^{|T|-|S|}g_{n,T}$$

对于$g_{n,S}$，其他点与S中点连边任意，因此其他点与S中点的连边方案数为$2^{|S|(n-|S|)}$，其他点之间的连边方案为$g_{n-|S|,\emptyset}$

$$g_{n,S}=2^{|S|(n-|S|)}g_{n-|S|,\emptyset}$$

因此

$$\begin{aligned}g_{n,\emptyset}&=\sum_{T\neq\emptyset}f_{n,T}=\sum_{T\neq\emptyset}\sum_{U\supseteq T}(-1)^{|U|-|T|}g_{n,U}\\&=\sum_{T\neq\emptyset}\sum_{U\supseteq T}(-1)^{|U|-|T|}2^{|U|(n-|U|)}g_{n-|U|,\emptyset}\\&=\sum_{U\neq\emptyset}(-1)^{|U|}2^{|U|(n-|U|)}g_{n-|U|,\emptyset}\sum_{\emptyset\neq T\subseteq U}(-1)^{|T|}\\&=\sum_{k=1}^{n}\binom{n}{k}(-1)^{k+1}2^{k(n-k)}g_{n-k,\emptyset}\end{aligned}$$

得到递推式

$$a_n=\sum_{k=1}^{n}\binom{n}{k}(-1)^{k+1}2^{k(n-k)}a_{n-k}$$

注：上述过程与思路过于奇妙深刻，暂时未能完全理解。

考虑优化，有

$$k(n-k)=kn-k^2=\frac{n^2+k^2-(n-k)^2}{2}-k^2=\frac{n^2}{2}-\frac{k^2}{2}-\frac{(n-k)^2}{2}$$

因此

$$\begin{aligned}a_n&=\sum_{k=1}^{n}\frac{n!}{k!(n-k)!}(-1)^{k+1}2^{\frac{n^2}{2}-\frac{k^2}{2}-\frac{(n-k)^2}{2}}a_{n-k}\\&=-2^{n^2/2}n!\sum_{k=1}^{n}\frac{(-1)^k2^{-k^2/2}}{k!}\frac{2^{-(n-k)^2/2}a_{n-k}}{(n-k)!}\\&=-2^{n^2/2}n!\sum_{k=0}^{n-1}\frac{(-1)^{n-k}2^{-(n-k)^2/2}}{(n-k)!}\frac{2^{-k^2/2}a_k}{k!}\end{aligned}$$

边界为$a_0=1$。

法1：使用分治卷积，在边界将系数乘上即可，复杂度$O(n\log^2 n)$。

法2：令

$$G(x)=\sum_{i=0}^{\infty}\frac{a_n}{2^{n^2/2}n!}x^i=\sum_{i=0}^{\infty}b_ix^i$$
$$H(x)=\sum_{i=0}^{\infty}\frac{(-1)^{n+1}2^{-n^2/2}}{n!}x^i=\sum_{i=0}^{\infty}c_ix^i$$

则有

$$n\geq 1\to b_n=\sum_{k=0}^{n-1}b_kc_{n-k}=\sum_{k=0}^{n}b_kc_{n-k}-b_nc_0=\sum_{k=0}^{n}b_kc_{n-k}+b_n$$
$$\sum_{k=0}^{n}b_kc_{n-k}=-[n=0]$$

因此$G(x)H(x)=-1$，求逆即可，复杂度$O(n\log n)$。

```
    vi f(n + 1);
    for (int i = 0; i <= n; ++i)
        f[i] = mul(mul(i & 1 ? 1: P - 1, inv(qpm(sqrt2, 1ll*i*i % (P - 1)))),
ifac[i]);
    vi g = pinv(f);
    for (int i = 0; i <= n; ++i)
        g[i] = sub(0, mul(g[i], mul(qpm(sqrt2,1ll * i * i % (P - 1)),
fac[i])));
    f = iegf(log(egf(g)));
```

有标号强连通图计数

设n个点的有标号有向图的数量为a_n，有标号强连通图数量为b_n，有标号DAG数量为c_n则

$$a_n=\sum_{k=1}^{n}c_k[x^n]\frac{G(x)^k}{k!}$$
$$F(x)=H(G(x))$$
$$G(x)=H^{-1}(F(x))$$

设n个点的有标号有向图的数量为a_n，有标号强连通图数量为b_n，有标号的每个弱连通分量都是强连通的有向图数量为c_n。

对于所有大小为n的有向图，设$f(S)$为有且仅有S中点都在出度为0的SCC中的方案数，$g(S)$为S中点组成了一些出度为0的SCC中的方案数，则有

确定了哪些点在出度为0的强连通分量中之后就可以将其他点与这些点随意连边。此时有

$$g(S)=\sum_{S\subseteq T}f(T)=2^{|S|(n-|S|)}c_{|S|}a_{n-|S|}$$

$$f(S)=\sum_{S\subseteq T}(-1)^{|T|-|S|}g(T)$$

$$\begin{aligned}a_n&=\sum_{S\subseteq V}f(S)=\sum_{S\subseteq V}\sum_{S\subseteq T}(-1)^{|T|-|S|}2^{|T|(n-|T|)}c_{|T|}a_{n-|T|}\\&=\sum_{k=0}^{n}\sum_{|T|=k}(-1)^k2^{k(n-k)}c_ka_{n-k}\sum_{S\subseteq T}(-1)^{|S|}\\&=\sum_{k=0}^{n}\binom{n}{k}(-1)^k2^{k(n-k)}c_ka_{n-k}[k=0]\end{aligned}$$

设n的点的强连通图数量的生成函数为$F(n)$，考虑用所有n个点的有向图的个数$H(n)$减去非强连通图的方案数来计算答案。

考虑DAG计数的过程，我们枚举非强连通图缩点后出度为0的强连通分量由哪些点组成，其他点与这些点之间可以任意连边。

可得：

$$F(n)=H(n)-\sum_{k=1}^{n}\binom{n}{k}2^{k(n-k)}H(n-k)G(k)+F(n)$$

其中$G(k)$表示k个点组成数个互不相连的强连通分量的方案数。

右边加入$F(n)$的原因是，当k取到n时，该项表示的方案为n个点组成数个互不相连的强连通分量的方案数，即恰好包括了一个$F(n)$，因此要将其加回去。

由DAG计数可知，因为对$H(n-k)$中的连边方案没有任何限制，所以余下$n-k$个点中的点在缩点后也可能产生新的0出度点，即上述计数过程会造成重复统计。

我们仿照DAG计数，令$G(n,i)$表示n个点组成i个互不相连的强连通分量的方案数，并令$G(n)=\sum_{i=1}^{k}(-1)^{i-1}G(n,i)$,加入容斥系数来避免重复统计。

由前式可得关于$G(n)$的递推式：

$$H(n)=\sum_{k=1}^{n}\binom{n}{k}2^{k(n-k)}H(n-k)G(k)$$

$$=\sum_{k=1}^{n-1}\binom{n}{k}2^{k(n-k)}H(n-k)G(k)+G(n)$$

于是

$$G(n)=H(n)-\sum_{k=1}^{n-1}\binom{n}{k}2^{k(n-k)}H(n-k)G(k)$$

考虑用$G(n)$获得$F(n)$的递推式，因为$F(n)=G(n,1)$，因此我们考虑通过枚举1号点所在强连通分量大小计算$\sum_{i=2}^{n}(-1)^iG(n,i)$然后加到$G(n)$上去。

$$F(n)=G(n,1)=G(n)+\sum_{i=2}^{n}(-1)^iG(n,i)$$

$$=G(n)+\sum_{i=2}^{n}(-1)^i\sum_{k=1}^{n-i+1}\binom{n-1}{k-1}G(k,1)G(n-k,i-1)$$

$$=G(n)+\sum_{k=1}^{n-1}\binom{n-1}{k-1}G(k,1)\sum_{i=1}^{n-k}(-1)^{i-1}G(n-k,i)$$

$$=G(n)+\sum_{k=1}^{n-1}\binom{n-1}{k-1}F(k)G(n-k)$$

注：$G(n,i)$可通过枚举1号点所在连通块大小获得递推式

$$G(n,i)=\sum_{k=1}^{n-i+1}\binom{n-1}{k-1}G(k,1)G(n-k,i)$$

有标号仙人掌计数

设n个点的有标号有根仙人掌数量的EGF为$F(x)$。

考虑一棵有根仙人掌的根，与其相邻的结构有两种：

1. 桥。桥对面恰好是一棵有根仙人掌。对应的生成函数即为$F(x)$。
2. 环。将环上的边删去后，环上的每一个点都是某棵仙人掌的根。对应的生成函数即为$\frac{1}{2}F(x)^{i-1}$。

将这些方案加起来后exp即得$n-1$个点的有根仙人掌数量。由此得到方程：

$$F(x)=x\exp\left(F(x)+\frac{1}{2}\sum_{i=2}^{\infty}F(x)^i\right)$$

$$=x\exp\left(F(x)+\frac{F^2(x)}{2-2F(x)}\right)=x\exp\left(\frac{2F(x)-F(x)^2}{2-2F(x)}\right)$$

考虑牛顿迭代：

$$t(F)=x\exp\left(\frac{2F-F^2}{2-2F}\right)-F$$

$$t'(F)=x\exp\left(\frac{2F-F^2}{2-2F}\right)\left(1+2\frac{2F-F^2}{(2-2F)^2}\right)-1$$

一大坨糊上去即可。

```
vi cactus(int n) {
    if (n == 1) { return { 0 }; }
    vi f = cactus0((n + 1) >> 1); f.resize(n);
    vi h1 = pinv(psub({2},pmul(f,2))), h2 = psub(pmul(f,2),pmul(f, f, n)), h3 = pmul(h2, h1, n), h4 = pmul(h3, h1, n);
    h3 = exp(h3); h3.insert(h3.begin(), 0); h3.pop_back();
    return psub(f, pmul(psub(h3, f), pinv(psub(pmul(h3, padd({1}, pmul(h4, 2)), n), {1})), n));
}
```

有标号点双连通图计数

设n个点的有根连通图的EGF为$F(x)$，n个点的点双连通图数量为g_n。

每个有根连通图的根都在几个点双中。对于其所在的某个点双，将属于这个点双中的边全部拆掉后剩下来的每个点都对应一个有根连通图。设这个点双大小为$i+1$，则该点双中的方案数对应的EGF为$g_{i+1}\dfrac{F(x)^i}{i!}$。

令$G(x)=\sum\limits_{i=1}^{\infty}g_{i+1}\dfrac{x^i}{i!}$，则有

$$F(x)=x\exp\left(\sum_{i=1}^{\infty}g_{i+1}\frac{F(x)^i}{i!}\right)=x\exp(G(F(x)))$$

将$F^{-1}(x)$代入x得

$$x=F^{-1}(x)\exp(G(x))$$
$$G(x)=\ln\frac{x}{F^{-1}(x)}$$

令$H(x)=\ln\frac{F(x)}{x}$，则有$G(x)=H(F^{-1}(x))$。

则$n+1$个点的带标号点双连通图个数为

$$n![x^n]G(x)=n!\frac{1}{n}[x^{n-1}]H(F^{-1}(x))=(n-1)![x^{n-1}]H'(x)\left(\frac{x}{F(x)}\right)^n$$
$$=(n-1)![x^{n-1}]H'(x)\left(\frac{1}{\exp(H(x))}\right)^n=(n-1)![x^{n-1}]H'(x)\exp(-nH(x))$$

```
int vbcc(int n) {
    n--;
    if (!n) return 1;
    int n1 = n + 2;
    vi f(n1, 0);
    f[0] = 1;
    for (int i = 1; i < n1; ++i)
        f[i] = qpm(2, (1ll * i * (i - 1) / 2) % (P - 1));
    f = log(egf(f));
    for (int i = 0; i < n1; ++i)
        f[i] = mul(f[i], i);
    f.erase(f.begin());
    vi h = log(f);
    vi hh = pmul(deriv(h), exp(pmul(h, sub(0, n))), n);
    return mul(fac[n - 1], hh[n - 1]);
}
```

有标号边双连通图计数

设n个点的有根连通图EGF为$F(x)$，n个点的有根边双连通图EGF为$G(x)$。

每个有根连通图的根都在某个有根边双中。对于其所在的边双，将属于这个边双的边全部拆掉后剩下来的每个点都连接着数个有根连通图。

设这个边双大小为i，则该边双的方案数为$g_i \dfrac{\exp(F(x))^i}{i!}$

总方案数为

$$F(x)=\sum_{i=1}^{\infty} g_i \frac{(x\exp(F(x)))^i}{i!}=G(x\exp(F(x)))$$

令$H(x)=x\exp(F(x))$

$$F(x)=G(H(x))$$

将$H^{-1}(x)$代入x得

$$G(x)=F(H^{-1}(x))$$

则n个点的有标号有根边双个数为

$$n![x^n]G(x)=n!\frac{1}{n}[x^{n-1}]F'(x)\left(\frac{x}{H(x)}\right)^n=(n-1)![x^{n-1}]F'(x)\exp(-nF(x))$$

```
int ebcc(int n) {
    int n1 = n + 1;
    vi f(n1, 0);
    f[0] = 1;
    for (int i = 1; i < n1; ++i)
        f[i] = qpm(2, (1ll * i * (i - 1) / 2) % (P - 1));
    f = log(egf(f));
    for (int i = 0; i < n1; ++i)
        f[i] = mul(f[i], i);
    vi hh = pmul(deriv(f), exp(pmul(f, sub(0, n))), n);
    return mul(invs[n], mul(fac[n - 1], hh[n - 1]));
}
```