7-1 封闭递推

生成函数一般形式为 $F(x)=\sum_{i=0}^{\infty}h_ix^i=h_0+h_1x+h_2x^2+\dots$

例1：求解下列数列的生成函数

(1) $a=1,1,1,1\dots$

$$F(x)=\sum_{n=0}^{\infty}x^n=1+x+x^2+\dots$$
$$F(x)x+1=F(x)$$
$$F(x)=\frac{1}{1-x}$$

(2) $a=1,p,p^2,p^3\dots$

$$F(x)px+1=F(x)$$
$$F(x)=\frac{1}{1-px}$$

(3) $a=0,1,1,1\dots$

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

(4) $a=1,0,1,0\dots$

$$F(x)=\sum_{n=0}^{\infty}x^{2n}=\frac{1}{1-x^2}$$

(5) $a=1,2,3,4\dots$

$$\begin{aligned}F(x)&=\sum_{n=0}^{\infty}(n+1)x^n\\&=\sum_{n=1}^{\infty}nx^{n-1}\\&=\sum_{n=1}^{\infty}(\int nx^{n-1}dx)'\\&=\sum_{n=1}^{\infty}(x^n)'\\&=(\frac{1}{1-x})'\\&=\frac{1}{(1-x)^2}\end{aligned}$$

(6) $a_n=\binom{m}{n}(m\text{是常数，}n\geq 0)$

$$F(x)=\sum_{n=0}^{\infty}\binom{m}{n}x^n=(1+x)^m(\text{二项式定理})$$

(7) $a_n=\binom{m+n}{n}(m\text{是常数，}n\geq 0)$

用数学归纳法，当$m=0$时，有$F(x)=\frac{1}{1-x}$。

当$m>0$时，有

$$\begin{aligned}\frac{1}{(1-x)^{m+1}} &= \frac{1}{(1-x)^m}\frac{1}{1-x} \\ &= (\sum_{n=0}^{\infty}\binom{m+n-1}{n}x^n)(\sum_{n=0}^{\infty}x^n) \\ &= \sum_{n=0}^{\infty}x^n\sum_{i=0}^{n}\binom{m+i-1}{i} \\ &= \sum_{n=0}^{\infty}\binom{m+n}{n}x^n\end{aligned}$$

7-2 生成函数

基本二项式公式

$$\sum_{k=0}^{n}x^k=\frac{1-x^{n+1}}{1-x}$$

$$\binom{-m}{n}=(-1)^n\binom{m+n-1}{n}$$

$$(1+x)^m=\sum_{n=0}^{\infty}(-1)^n\binom{m+n-1}{n}x^n$$

$$(1-x)^m=\sum_{n=0}^{\infty}\binom{m+n-1}{n}x^n$$

$$(\frac{1}{1-x})^n=\sum_{k=0}^{\infty}\binom{-n}{k}(-x)^k,\binom{-n}{k}=(-1)^k\binom{n+k-1}{k}$$

$$(\frac{1}{1-x})^n=\sum_{m=0}^{\infty}\binom{n+m-1}{m}x^m$$

例2：P10780 食物

每种食物的限制如下：

- **承德汉堡：偶数个；**
- **可乐：0 个或 1 个；**
- **鸡腿：0 个，1 个或 2 个；**
- **蜜桃多：奇数个；**
- **鸡块：4 的倍数个；**
- **包子：0 个，1 个，2 个或 3 个；**
- **土豆片炒肉：不超过一个；**
- **面包：3 的倍数个；**

对于给出的 n，你需要计算出方案数，并对 10007 取模。

本题的生成函数很容易推出来，为$\frac{x}{(1-x)^4}$，难点在于如何展开。

$$
\begin{aligned}
\frac{x}{(1-x)^4} &= x(1-x)^{-4} \\
&= x\sum_{k=0}^{\infty}\binom{-4}{k}x^k(-1)^{-4-k} \\
&= x\sum_{k=0}^{\infty}(-1)^k\binom{k+3}{k}x^k(-1)^{-4-k} \\
&= x\sum_{k=0}^{\infty}\binom{k+3}{k}x^k \\
&= \sum_{k=1}^{\infty}\binom{k+2}{k-1}x^k \\
&= \sum_{k=1}^{\infty}\frac{(k+2)(k+1)k}{6}x^k
\end{aligned}
$$

设$F(x),G(x)$分别为长度为 k 的序列 a_n,b_n 的生成函数，则

$$F(x)+G(x)=\sum_{n=0}^{k}(a_n+b_n)x^n$$

$$F(x)G(x)=(\sum_{n=0}^{k}a_nx^n)(\sum_{n=0}^{k}b_nx^n)=\sum_{n=0}^{k}x^n\sum_{i=0}^{n}a_ib_{n-i}$$

例3：Devu and Flowers

Devu 想用花去装饰他的花园，他已经购买了n个箱子，第i个箱子有f_i朵花，在同一个的箱子里的所有花是同种颜色的（所以它们没有任何其他特征）。另外，不存在两个箱子中的花是相同颜色的。

现在 Devu 想从这些箱子里选择s朵花去装饰他的花园，Devu 想要知道，总共有多少种方式从这些箱子里取出这么多的花？因为结果有可能会很大，结果需要对10^9+7取模。 Devu 认为至少有一个箱子中选择的花的数量不同才是两种不同的方案。$(1\le n\le 20, 0\le s\le 10^{14}, 0\le f_i\le 10^{12})$

显然的是，本题生成函数为

$$F(x)=\prod_{i=1}^{n}\sum_{k=0}^{f_i}x^k=\prod_{i=1}^{n}\frac{1-x^{f_i+1}}{1-x}=(\frac{1}{1-x})^n\prod_{i=1}^{n}(1-x^{f_i+1})$$

由于 n 只有 20 ，因此右边部分的式子可以进行 2^n 的暴力枚举，对于左边的式子，通过其在方程解的个数上的意义可得

$$(\frac{1}{1-x})^n=\sum_{m=0}^{\infty}\binom{n+m-1}{m}x^m$$

总时间复杂度为 $O(2^n\times n)$，代码如下：

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll mod=1e9+7;
ll fac=1,n,f[100],s,ans=0;
ll fastpow(ll a,ll k)
{
    ll res=1;
    a%=mod;
    while(k)
```

```
    {
        if(k&1)  res=res*a%mod;
        k>>=1;
        a=a*a%mod;
    }
    return res;
}
ll C(ll k)
{
    ll res=1;
    for(ll i=s-k+1;i<=n+s-k-1;i++)  res=res*(i%mod)%mod;
    return res*fastpow(fac,mod-2)%mod;
}
void dfs(int pos,ll opt,ll k)//位置、选择1或-x、x的k次方
{
    if(pos==n)
    {
        if(k>s)  return;
        ans=((ans+opt*C(k))%mod+mod)%mod;
        return;
    }
    dfs(pos+1,opt,k);
    dfs(pos+1,-opt,k+f[pos+1]+1);
}
int main()
{
    cin>>n>>s;
    for(ll i=1;i<n;i++)  fac*=i;
    for(int i=1;i<=n;i++)  cin>>f[i];
    dfs(0,1,0);
    cout<<ans;
}
```

7-3 指数型生成函数

指数型生成函数的形式为 $\widehat{F(x)}=\sum_{i=0}^{\infty}h_i\frac{x^n}{n!}$，多用于组合意义。

序列 $a=0,1,1,1...$ 的生成函数为 $\widehat{F(x)}=\sum_{i=0}^{\infty}\frac{x^n}{n!}=e^x$。

例4：Blocks

一段长度为 n 的序列，你有红黄蓝绿 4 种颜色的砖块，一块砖长度为 1，问你铺砖的方案数，其中红色砖和绿色砖的数量必须为偶数。答案可能很大，请输出 $mod\ 10007$ 后的结果。

四种颜色中，有两种颜色的数量需要为偶数，生成函数为

$$\begin{aligned}\widehat{F(x)}&=(1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+...)^2(1+\frac{x^2}{2!}+\frac{x^4}{4!}+...)^2\\&=(\sum_{n=0}^{\infty}\frac{x^n}{n!})^2(\sum_{n=0}^{\infty}\frac{x^{2n}}{(2n)!})^2\\&=(e^x)^2(\frac{e^x+e^{-x}}{2})^2\\&=\frac{1}{4}(e^{4x}+2e^{2x}+1)\end{aligned}$$

对 e^x 进行泰勒展开，得到通项公式为

$$h_n = \frac{1}{4}(4^{n-1} + 2^{n-1})$$

7-4 常系数齐次线性递推

例5：斐波那契数列$f_n = f_{n-1} + f_{n-2}, f_0 = 0, f_1 = 1$

设斐波那契数列的生成函数为$G(x)$，令递推式一边为0，得到$f_n - f_{n-1} - f_{n-2} = 0$，其特征方程为$r^2 - r - 1 = 0$，两个实根分别为$r_1 = \frac{1+\sqrt{5}}{2}, r_2 = \frac{1-\sqrt{5}}{2}$，开始推导其生成函数：

$$\begin{aligned} G(x) &= h_0 + h_1x + h_2x^2 + h_3x^3 + \dots \\ -xG(x) &= \quad\ - h_0x - h_1x^2 - h_2x^3 + \dots \\ -x^2G(x) &= \qquad\qquad - h_0x^2 - h_1x^3 + \dots \end{aligned}$$

上面三个式子相加得到

$$(1 - x - x^2)G(x) = h_0 + (h_1 - h_0)x + (h_2 - h_1 - h_0)x^2 + (h_3 - h_2 - h_1)x^3 + \dots$$

由递推式可知，$h_n - h_{n-1} - h_{n-2} = 0, h_0 = 0, h_1 = 1$，故上式等于

$$\begin{aligned} (1 - x - x^2)G(x) = h_0 + (h_1 - h_0)x = x \\ G(x) = \frac{x}{1 - x - x^2} \end{aligned}$$

我们得到了斐波那契数列的生成函数$G(x)$，其分母转化为特征方程后就可以找到递推式，分子与递推式的初值有关。在一开始我们得知该特征方程共有两个不同的实根，因此用待定系数法先将$G(x)$拆成两个有理分式以便进行二项式展开：

$$\begin{aligned} G(x) &= \frac{x}{1 - x - x^2} \\ &= -\left(\frac{c_1}{x - \frac{1+\sqrt{5}}{2}} + \frac{c_2}{x - \frac{1-\sqrt{5}}{2}}\right) \\ &= -\frac{c_1(x - \frac{1-\sqrt{5}}{2}) + c_2(x - \frac{1+\sqrt{5}}{2})}{x^2 - x - 1} \end{aligned}$$

列出方程

$$\begin{cases} c_1 + c_2 = 1 \\ -\frac{1-\sqrt{5}}{2}c_1 - \frac{1+\sqrt{5}}{2}c_2 = 0 \end{cases}$$

解得

$$c_1 = \frac{1+\sqrt{5}}{2\sqrt{5}}, c_2 = \frac{\sqrt{5}-1}{2\sqrt{5}}$$

代入回生成函数得

$$
\begin{aligned}
G(x) &= -(\frac{c_1}{x-\frac{1+\sqrt{5}}{2}}+\frac{c_2}{x-\frac{1-\sqrt{5}}{2}}) \\
&= \frac{\frac{1+\sqrt{5}}{2\sqrt{5}}}{\frac{1+\sqrt{5}}{2}-x}+\frac{\frac{\sqrt{5}-1}{2\sqrt{5}}}{\frac{1-\sqrt{5}}{2}-x} \\
&= \frac{\frac{1}{\sqrt{5}}}{1-\frac{\sqrt{5}-1}{2}x}+\frac{\frac{-1}{\sqrt{5}}}{1-\frac{1+\sqrt{5}}{-2}x} \\
&= \frac{1}{\sqrt{5}}\sum_{n=0}^{\infty}(\frac{\sqrt{5}-1}{2})^n x^n-\frac{1}{\sqrt{5}}\sum_{n=0}^{\infty}(\frac{\sqrt{5}+1}{-2})^n x^n
\end{aligned}
$$

可得通项公式为

$$f_n=\frac{1}{\sqrt{5}}((\frac{\sqrt{5}+1}{2})^n-(\frac{\sqrt{5}-1}{2})^n)$$

例6：P4451 [国家集训队] 整数的lqp拆分

lqp在为出题而烦恼，他完全没有头绪，好烦啊...

他首先想到了整数拆分。整数拆分是个很有趣的问题。给你一个正整数 N ，对于N的一个整数拆分就是满足任意 $m>0$，$a_1,a_2,a_3\dots a_m>0$，且 $a_1+a_2+a_3+\dots+a_m=n$ 的一个有序集合。通过长时间的研究我们发现了计算对于 n 的整数拆分的总数有一个很简单的递推式，但是因为这个递推式实在太简单了，如果出这样的题目，大家会对比赛毫无兴趣的。

然后 lqp 又想到了斐波那契数。定义 $F_0=0,F_1=1,F_n=F_{n-1}+F_{n-2}(n>1)$，$F_n$就是斐波那契数的第 n 项。但是求出第 n 项斐波那契数似乎也不怎么困难...

lqp 为了增加选手们比赛的欲望，于是绞尽脑汁，想出了一个有趣的整数拆分，我们暂且叫它：整数的lqp拆分。

和一般的整数拆分一样，整数的 lqp 拆分是满足任意 $m>0$，$a_1,a_2,a_3\dots a_m>0$，且 $a_1+a_2+a_3+\dots+a_m=n$ 的一个有序集合。但是整数的lqp拆分要求的不是拆分总数，相对更加困难一些。

对于每个拆分，lqp 定义这个拆分的权值 $F_{a_1},F_{a_2}\dots F_{a_m}$，他想知道对于所有的拆分，他们的权值之和是多少?

简单来说，就是求
$\sum\prod_{i=1}^{m}F_{a_i}$

$m>0$

$a_1,a_2,\dots,a_m>0$

$a_1+a_2+\dots+a_m=n$
由于答案可能非常大，所以要对 10^9+7 取模。

$n\le 10^{10000}$**，时间限制**$1000ms$**。**

我们在上一个例子中得到了斐波那契数列的生成函数，在本题中若拆分为 k 个数，连乘部分的生成函数为 $F(x)^k$，答案的生成函数为

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

由 $G(x)$ 可知递推式为 $h_n=2h_{n-1}+h_{n-2},h_0=0,h_1=1$，此时可以用矩阵快速幂做，但是时间很紧，而且还需要用高精度，可以继续推导通项公式。特征方程的两个根为 $r_1=-1+\sqrt{2},r_2=-1-\sqrt{2}$。经推导通项公式为

$$h(n)=\frac{1}{2\sqrt{2}}((1+\sqrt{2})^n-(1-\sqrt{2})^n)\%1000000007$$

此时我们需要求出 $\sqrt{2}$ 取模后的结果，可以进行打表，设 i 为答案，那么 $i\times i\equiv 2(mod\ 1e9+7)$，打表后得知 $i=59713600$。

解决了根号取模的问题，我们还需要考虑本题 n 的超大范围。根据费马小定理，$a^{p-1}\equiv 1(mod\ p)$，故 $a^n\%p=a^{n\%(p-1)}$。

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll mod=1e9+7,p=59713600;
ll fastpow(ll a,ll k)
{
    ll res=1;
    a%=mod;
    while(k)
    {
        if(k&1)  res=res*a%mod;
        k>>=1;
        a=a*a%mod;
    }
    return res;
}
ll inv(ll x)
{
    return fastpow(x,mod-2);
}
int main()
{
    string s;
    ll n=0;
    cin>>s;
    for(int i=0;i<s.length();i++)  n=(n*10+s[i]-'0')%(mod-1);
    ll c=inv(2)*inv(p)%mod,y=fastpow(p-1+mod,n),z=fastpow(-p-1+mod,n);
    ll ans=c*(y-z+mod)%mod;
    if(n%2==0)  ans=mod-ans;
    cout<<ans;
}
```

7-5 快速傅里叶变换

一些情况下，我们得到的生成函数不能轻易展开，需要用到FFT来计算卷积，以得到多项式中的某个系数。

例7：TSUM - Triple Sums

给定一个由 N 个不同整数组成的序列 s。
考虑序列中不同索引的三个整数的所有可能的和。
对于每个可能的和，输出生成它的不同递增索引三元组的数量。

$3\le N\le 40000$，$s_i(0\le |s_i|\le 20000)$，**保证没有两个整数相同。**

我们可以得知，答案的生成函数为$F(x)=(1+x^{a_1}+x^{a_2}+\dots)^3$，不过这样会导致我们选择的三元组中含有相同索引，需要利用容斥。

设$G(x)$为三元组中出现了两个相同索引的生成函数，$H(x)$为三元组中三个索引全相同的生成函数。则$G(x)=(1+x^{2a_1}+x^{2a_2}+\dots)^2(1+x^{a_1}+x^{a_2}+\dots)$，$H(x)=(1+x^{3a_1}+x^{3a_2}+\dots)$。

首先，去除形如$(i,i,j),(i,j,i),(i,i,j)$这 3 种三元组，令 $F(x)$ 减去 $3G(x)$，就去除了出现两个相同索引的情况。

然而，这样的话，我们发现，$F(x)$ 与 $G(x)$ 中均包含了 1 次三个索引都相同情况，而 $F(x)-3G(x)$ 相当于变化了 $(1-3)=-2$ 次三中索引都相同的情况。因此，结果还需要加上 $2H(x)$，最终答案为 $F(x)-3G(x)+2H(x)$。

由于我们前面是按照排列的方式来做的，转化为组合时应注意将系数除以 6。

```
#include <bits/stdc++.h>
using namespace std;
#define SZ(x) ((int)((x).size()))
#define lb(x) ((x) & (-(x)))
#define bp(x) __builtin_popcount(x)
#define bpll(x) __builtin_popcountll(x)
#define mkp make_pair
#define pb push_back
#define fi first
#define se second
typedef long long ll;
typedef unsigned long long ull;
typedef pair<int, int> pii;
typedef pair<ll, int> pli;
typedef pair<ll, ll> pll;
typedef pair<double, int> pdi;
const int N=2e4;
namespace Polynomial {
    const int Mod = 998244353;
    const int G = 3;

    template<int mod>
    class Modint {
        private:
            unsigned int x;
        public:
            Modint() = default;
            Modint(unsigned int x): x(x) {}
        friend istream& operator >> (istream& in, Modint& a) {return in >> a.x;}
        friend ostream& operator << (ostream& out, Modint a) {return out <<
a.x;}
        friend Modint operator + (Modint a, Modint b) {return (a.x + b.x) %
mod;}
        friend Modint operator - (Modint a, Modint b) {return (a.x - b.x + mod) %
mod;}
        friend Modint operator * (Modint a, Modint b) {return 1ULL * a.x * b.x %
mod;}
        friend Modint operator / (Modint a, Modint b) {return a * ~b;}
        friend Modint operator ^ (Modint a, int b) {Modint ans = 1; for(; b; b
>>= 1, a *= a) if(b & 1) ans *= a; return ans;}
        friend Modint operator ~ (Modint a) {return a ^ (mod - 2);}
        friend Modint operator - (Modint a) {return (mod - a.x) % mod;}
        friend Modint& operator += (Modint& a, Modint b) {return a = a + b;}
```

```
        friend Modint& operator -= (Modint& a, Modint b) {return a = a - b;}
        friend Modint& operator *= (Modint& a, Modint b) {return a = a * b;}
        friend Modint& operator /= (Modint& a, Modint b) {return a = a / b;}
        friend Modint& operator ^= (Modint& a, int b) {return a = a ^ b;}
        friend Modint& operator ++ (Modint& a) {return a += 1;}
        friend Modint operator ++ (Modint& a, int) {Modint x = a; a += 1; return
x;}
        friend Modint& operator -- (Modint& a) {return a -= 1;}
        friend Modint operator -- (Modint& a, int) {Modint x = a; a -= 1; return
x;}
        friend bool operator == (Modint a, Modint b) {return a.x == b.x;}
        friend bool operator != (Modint a, Modint b) {return !(a == b);}
    };
    typedef Modint<Mod> mint;

    void init_convo(vector<int>& rev, int& lim, int n, int m) {
        int s = 0;
        for (lim = 1; lim <= n + m; lim <<= 1, s++);
        rev.resize(lim);
        for (int i = 0; i < lim; i++) {
            rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (s - 1));
        }
    }

    void NTT(vector<mint>& f, vector<int>& rev, int lim, int type) {
        f.resize(lim, 0);
        for (int i = 0; i < lim; i++) {
            if (i < rev[i]) {
                swap(f[i], f[rev[i]]);
            }
        }
        for (int i = 1; i < lim; i <<= 1) {
            mint mul = (mint)G ^ ((Mod - 1) / (i << 1));
            if (type == -1) {
                mul = ~mul;
            }
            for (int j = 0; j < lim; j += (i << 1)) {
                mint w = 1;
                for (int k = 0; k < i; k++, w *= mul) {
                    mint x = f[j + k], y = w * f[j + k + i];
                    f[j + k] = x + y;
                    f[j + k + i] = x - y;
                }
            }
        }
    }

    vector<mint> convolution(vector<mint> f, vector<mint> g, int k1, int k2, mint
(*calc)(mint x, mint y)) {
        int n = SZ(f) - 1, m = SZ(g) - 1, lim;
        vector<int> rev;
        init_convo(rev, lim, k1 * n, k2 * m);

        NTT(f, rev, lim, 1);
        NTT(g, rev, lim, 1);
        vector<mint> h(lim);
        for (int i = 0; i < lim; i++) {
            h[i] = calc(f[i], g[i]);
```

```
    }
    NTT(h, rev, lim, -1);
    mint invlim = ~(mint)lim;
    for (int i = 0; i < lim; i++) {
        h[i] = h[i] * invlim;
    }
    h.resize(k1 * n + k2 * m + 1);
    return h;
}
vector<mint> convolution(const vector<mint>& f, const vector<mint>& g) {
    return convolution(f, g, 1, 1, [](mint x, mint y) -> mint {
        return x * y;
    });
}

vector<mint> derivation(vector<mint> f) {
    int n = SZ(f);
    for (int i = 1; i < n; i++) {
        f[i - 1] = f[i] * i;
    }
    f.resize(n - 1);
    return f;
}
vector<mint> integrate(vector<mint> f) {
    int n = SZ(f);
    f.resize(n + 1, 0);
    for (int i = n - 1; i >= 0; i--) {
        f[i + 1] = f[i] / (i + 1);
    }
    f[0] = 0;
    return f;
}

vector<mint> polyadd(const vector<mint>& f, const vector<mint>& g) {
    int n = SZ(f), m = SZ(g);
    int mx = max(n, m), mn = min(n, m);
    vector<mint> h(mx);
    for (int i = 0; i < mn; i++) {
        h[i] = f[i] + g[i];
    }
    for (int i = mn; i < mx; i++) {
        h[i] = (n > m) ? f[i] : g[i];
    }
    return h;
}
vector<mint> polysub(const vector<mint>& f, const vector<mint>& g) {
    int n = SZ(f), m = SZ(g);
    int mx = max(n, m), mn = min(n, m);
    vector<mint> h(mx);
    for (int i = 0; i < mn; i++) {
        h[i] = f[i] - g[i];
    }
    for (int i = mn; i < mx; i++) {
        h[i] = (n > m) ? f[i] : -g[i];
    }
    return h;
}
```

```
    vector<mint> polyinv(vector<mint> f, int n) {   // 1 / f(x) (mod x^n)
        f.resize(n);
        if (n == 1) {
            f[0] = ~f[0];
            return f;
        }
        auto g = polyinv(f, (n + 1) >> 1);
        g = convolution(f, g, 1, 2, [](mint x, mint y) -> mint {
            return (2 - x * y) * y;
        });
        g.resize(n, 0);
        return g;
    }

    vector<mint> polyln(vector<mint> f, int n) {    // ln f(x) (mod x^n) , f[0] =
1
        f = integrate(convolution(derivation(f), polyinv(f, n)));
        f.resize(n, 0);
        return f;
    }

    vector<mint> polyexp(vector<mint> f, int n) {   // exp f(x) (mod x^n) , f[0]
= 0 | Newton's Method
        f.resize(n, 0);
        if (n == 1) {
            f[0] = 1;
            return f;
        }
        auto g0 = polyexp(f, (n + 1) >> 1);
        auto g1 = polysub(polyln(g0, n), f);
        auto h = convolution(g0, g1, 1, 1, [](mint x, mint y) -> mint {
            return x * (1 - y);
        });
        h.resize(n, 0);
        return h;
    }

    vector<mint> __polyksm(vector<mint> f, int k, int n) {    // [f(x)]^{k} (mod
x^n) , f[0] = 0 | exp(ln f(x))
        f = polyln(f, n);
        for (int i = 0; i < n; i++) {
            f[i] *= k;
        }
        f = polyexp(f, n);
        f.resize(n, 0);
        return f;
    }

    vector<mint> polyksm(vector<mint> f, int k, int n) {    // [f(x)]^{k} (mod
x^n)
        f.resize(n, 0);
        int p = 0;
        for (int i = 0; i < n; i++) {
            if (f[i] != 0) {
                p = i;
                break;
            }
        }
```

```
        mint coef = ~f[p];
        vector<mint> g(n - p);
        for (int i = 0; i < n - p; i++) {
            g[i] = f[i + p] * coef;
        }
        g = __polyksm(g, k, n);

        ll d = 1ll * p * k;
        coef = (~coef) ^ k;
        fill(f.begin(), f.end(), 0);
        for (int i = n - 1; i >= d; i--) {
            f[i] = g[i - d] * coef;
        }
        return f;
    }
}
using Polynomial::convolution;
using Polynomial::Mod;
using Polynomial::mint;
int main()
{
    int n,mx=0,a[40010];
    cin>>n;
    for(int i=1;i<=n;i++)
    {
        cin>>a[i];
        a[i]+=N;
        mx=max(mx,a[i]);
    }
    vector<mint>f(mx+10,0),g(2*mx+10,0),h(3*mx+10,0);
    for(int i=1;i<=n;i++)
    {
        f[a[i]]++;
        g[2*a[i]]++;
        h[3*a[i]]++;
    }
    auto F=convolution(f,convolution(f,f));
    auto G=convolution(f,g);
    for(int i=1;i<=3;i++)  F=polysub(F,G);
    for(int i=1;i<=2;i++)  F=polyadd(F,h);
    for(int i=0;i<F.size();i++)
        if(F[i]!=0)  cout<<i-3*N<<" : "<<F[i]/6<<"\n";
}
```