

# 多项式 4

## 循环卷积

### 定义

$$c_k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} [i+j \bmod p=k] a_i b_j$$

### 性质

对序列 \$A, B\$ 做长度为 \$n\$ 的 \$\text{FFT}\$ 等价于求序列 \$A, B\$ 的长度为 \$n\$ 的循环卷积。

考虑单位根反演证明

$$\begin{aligned} c_k &= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} [i+j \bmod p=k] a_i b_j \\ &= \frac{1}{n} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{d=0}^{n-1} w_n^{d(i+j-k)} a_i b_j \\ &= \frac{1}{n} \sum_{d=0}^{n-1} w_n^{-dk} \left( \sum_{i=0}^{n-1} a_i w_n^{di} \right) \left( \sum_{j=0}^{n-1} b_j w_n^{dj} \right) \end{aligned}$$

不难发现 \$\left( \sum\_{i=0}^{n-1} a\_i w\_n^{di} \right) \left( \sum\_{j=0}^{n-1} b\_j w\_n^{dj} \right)\$ 即为原来的 \$\text{DFT}\$ 过程，\$\frac{1}{n} \sum\_{d=0}^{n-1} w\_n^{-dk}\$ 即为原来的 \$\text{IDFT}\$ 证毕。

事实上普通的卷积计算相当于长度为 \$2^n\$ 的循环卷积计算，只是循环卷积长度大于 \$C\$ 序列的长度，所以循环卷积结果即为普通卷积结果。

值得一提的是，循环卷积的求逆、快速幂等操作直接对 \$\text{DFT}\$ 的点值进行相应运算即可。

另外注意到 \$f(w^{-k}) = f(w^{n-k})\$ 于是循环卷积的 \$\text{IDFT}\$ 过程可以通过将序列 \$[1, n-1]\$ 部分翻转再 \$\text{DFT}\$ 的方式实现。

## Cooley-Tukey FFT algorithm

### 算法实现

普通 \$\text{DFT}\$ 过程是将序列根据奇偶幂次分成两段经行递归，现考虑将序列分成 \$d\$ 段进行递归，设

$$f(x) = a_0 + a_1 x^1 + a_2 x^2 + \dots + a_{n-1} x^{n-1}, f_k(x) = a_k x^k + a_{d+k} x^{d+k} + \dots + a_{n-d+k} x^{n-d+k} \quad (0 \le k < d), m = \frac{n}{d}$$

于是有

$$f(w_n^{im+j}) = \sum_{k=0}^{d-1} w_n^{(im+j)k} f_k(w_n^{(im+j)d}) = \sum_{k=0}^{d-1} w_n^{(im+j)k} f_k(w_n^{in} w_n^{jd}) = \sum_{k=0}^{d-1} w_n^{(im+j)k} f_k(w_m^j)$$

计算出每个点值的时间为  $O(d)$  每层共  $O(n)$  个点值。设  $n=p_1^{\alpha_1}p_2^{\alpha_2}\cdots p_k^{\alpha_k}$  于是总时间复杂度  $O\left(n\sum_{i=1}^k p_i^{\alpha_i}\right)$

## 算法例题

### 洛谷4191

给定长度为  $n$  的序列  $A, B$  计算  $AB^C$  的长度为  $n$  的循环卷积模  $n+1$  意义下的值。

数据保证  $n+1$  为素数  $n \leq 5 \times 10^5$   $n$  的最大质因子不超过  $10$ 。

### 题解

$\text{Cooley-Tukey FFT algorithm}$  板子题  $\text{DFT}$  后直接对点值快速幂即可，时间复杂度  $O(7n \log n)$

```
const int MAXN=5e5+5,MAXM=20,MAX_div=7;
int p;
int quick_pow(int a,int b){
    int ans=1;
    while(b){
        if(b&1)ans=1LL*ans*a%p;
        a=1LL*a*a%p;
        b>>=1;
    }
    return ans;
}
vector<int> pdiv,frac,Wn[MAXM];
bool check(int x){
    _for(i,0,pdiv.size()){
        if(quick_pow(x,(p-1)/pdiv[i])==1)
            return false;
    }
    return true;
}
void get_G(int n){
    int temp=p-1,g;
    for(int i=2;i*i<=temp;i++){
        if(temp%i==0){
            pdiv.push_back(i);
            while(temp%i==0)temp/=i;
        }
    }
    if(temp!=1)pdiv.push_back(temp);
    _for(i,2,p){
        if(check(i)){
            g=i;
        }
    }
}
```

```

        break;
    }
}
temp=n;
for(int i=2;i*i<=temp;i++){
    while(temp%i==0){
        frac.push_back(i);
        temp/=i;
    }
}
if(temp!=1)frac.push_back(temp);
int len=n,wn;
_for(i,0,frac.size()){
    Wn[i].resize(len);
    wn=quick_pow(g,(p-1)/len),Wn[i][0]=1;
    _for(j,1,len)Wn[i][j]=1LL*Wn[i][j-1]*wn%p;
    len/=frac[i];
}
}
int temp[MAXN];
void DFT(int *f,int n,int dep=0){
    if(n==1)return;
    memcpy(temp,f,sizeof(int)*n);
    int m=n/frac[dep],d=frac[dep],*g[MAX_div];
    _for(i,0,d)g[i]=f+i*m;
    for(int i=0,k=0;i<n;i+=d,k++){
        _for(j,0,d)
            g[j][k]=temp[i+j];
    }
    _for(i,0,d)DFT(g[i],m,dep+1);
    _for(i,0,d){
        _for(j,0,m){
            int pos=i*m+j;
            temp[pos]=0;
            _for(k,0,d)
                temp[pos]=(temp[pos]+1LL*Wn[dep][pos*k%n]*g[k][j])%p;
        }
    }
    memcpy(f,temp,sizeof(int)*n);
}
void IDFT(int *f,int n){
    reverse(f+1,f+n);
    DFT(f,n);
    int div=quick_pow(n,p-2);
    _for(i,0,n)f[i]=1LL*f[i]*div%p;
}
int a[MAXN],b[MAXN];
int main()
{
    int n=read_int(),k=read_int();
    p=n+1;
}

```

```
get_G(n);  
_for(i,0,n)a[i]=read_int();  
_for(i,0,n)b[i]=read_int();  
DFT(a,n);DFT(b,n);  
_for(i,0,n)a[i]=1LL*a[i]*quick_pow(b[i],k)%p;  
IDFT(a,n);  
_for(i,0,n)enter(a[i]);  
return 0;  
}
```

## Bluestein's Algorithm

### 算法实现

考虑  $\text{DFT}$  过程，有

$$\begin{aligned} y_k &= \sum_{i=0}^{n-1} a_i w_n^{ki} \\ &= \sum_{i=0}^{n-1} a_i w_n^{\binom{k+i}{2} - \binom{k}{2} - \binom{i}{2}} \\ &= w_n^{-\binom{k}{2}} \sum_{i=0}^{n-1} \left( a_i w_n^{\binom{i}{2}} \right) w_n^{\binom{k+i}{2}} \end{aligned}$$

其中  $\binom{n}{2}$  表示组合数，易知上式可以通过普通卷积求解。

考虑  $\text{IDFT}$  过程，同样有

$$\begin{aligned} a_k &= \frac{1}{n} \sum_{i=0}^{n-1} y_i w_n^{-ki} \\ &= \frac{1}{n} \sum_{i=0}^{n-1} y_i w_n^{-\binom{k+i}{2} + \binom{k}{2} + \binom{i}{2}} \\ &= \frac{1}{n} w_n^{\binom{k}{2}} \sum_{i=0}^{n-1} \left( y_i w_n^{\binom{i}{2}} \right) w_n^{-\binom{k+i}{2}} \end{aligned}$$

### 算法例题

#### 洛谷p5293

给定一个二维  $[L+1] \times n$  的空间，其中  $(u_1, v_1) \rightarrow (u_2, v_2)$  有  $w_{v_1, v_2}$  条重边。

假设起点为  $(0, x)$  终点为  $(\text{last}, y)$  ( $\text{last}$  为任意值)，路径长度  $m$  定义为路径的边数。

对每个  $0 \leq t \leq k$  求满足所有  $m \equiv t \pmod k$  且横坐标单增的路径数模  $p$  意义下的值。

数据保证  $p$  为素数  $10^8 \leq p \leq 2^{30}$ ,  $k \mid p-1$ ,  $1 \leq k \leq 65536$ ,  $1 \leq n \leq 3$ ,  $L \leq 10^9$

### 题解

假设  $f_{a,b}$  表示  $m=a$  且  $y=b$  的路径数  $g_{a,b}$  表示将空间的  $X$  维消去后  $m=a$  且  $y=b$  的路径数。

于是有状态转移方程

$$g_{a,b} = \sum_{i=1}^n g_{a-1,i} w_{i,b} \text{ , } f_{a,b} = \binom{L}{a} g_{a,b}$$

设矩阵

$$W = \begin{bmatrix} w_{1,1} & \cdots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{n,1} & \cdots & w_{n,n} \end{bmatrix}$$

设  $G_i = (g_{i,1} \cdots g_{i,n})$  于是有  $G_i = G_0 W^i$

考虑单位根反演，设  $w_k \equiv g^{\frac{(p-1)k}{p}} \pmod p$ ,  $g$  为  $p$  的原根，有

$$\begin{aligned} \sum_{i=0}^L f_{i,y} w_k^{\lfloor i \rfloor \bmod k} &= \frac{1}{k} \sum_{i=0}^L f_{i,y} \sum_{j=0}^{k-1} w_k^{\lfloor i \rfloor j} \\ &= \frac{1}{k} \sum_{j=0}^{k-1} w_k^{-\lfloor j \rfloor} \sum_{i=0}^L f_{i,y} w_k^{\lfloor i \rfloor j} \end{aligned}$$

根据二项式定理，有

$$\sum_{i=0}^L w_k^{\lfloor i \rfloor j} (f_{i,1} \cdots f_{i,n}) = \sum_{i=0}^L w_k^{\lfloor i \rfloor j} \binom{L}{i} (g_{i,1} \cdots g_{i,n}) = G_0 \sum_{i=0}^L \binom{L}{i} (w_k^j)^i = G_0 (w_k^j + 1)^L$$

于是根据矩阵快速幂可以  $O(kn^3 \log L)$  计算出所有  $h_j = \sum_{i=0}^L f_{i,y} w_k^{\lfloor i \rfloor j} (0 \leq j < k)$

于是有

$$\text{ans}_t = \frac{1}{k} \sum_{i=0}^{k-1} w_k^{-ti} h_i$$

发现上式就是  $\text{Bluestein's Algorithm}$  的  $\text{IDFT}$  过程，直接求解时间复杂度  $O(k \log k)$

总时间复杂度  $O(kn^3 \log L)$  主要用于计算矩阵快速幂。

```

const int MAXN=1<<16|5;
const long double pi=acos(-1.0);
int p,sz,gw[MAXN];
int quick_pow(int a,int b){
    int ans=1;
    while(b){
        if(b&1)ans=1LL*ans*a%p;
        a=1LL*a*a%p;
        b>>=1;
    }
    return ans;
}
vector<int> pdiv;
bool check(int x){
    _for(i,0,pdiv.size()){
        if(quick_pow(x,(p-1)/pdiv[i])==1)
            return false;
    }
    return true;
}
void get_G(int k){
    int temp=p-1,g;
    for(int i=2;i*i<=temp;i++){
        if(temp%i==0){
            pdiv.push_back(i);
            while(temp%i==0)temp/=i;
        }
    }
}

```

```
    }  
}  
if(temp!=1)pdiv.push_back(temp);  
_for(i,2,p){  
    if(check(i)){  
        g=i;  
        break;  
    }  
}  
int w=quick_pow(g,(p-1)/k);  
gw[0]=1;  
_rep(i,1,k)gw[i]=1LL*gw[i-1]*w%p;  
}  
struct Matrix{  
    int ele[3][3];  
    Matrix(int x=0){  
        mem(ele,0);  
        _for(i,0,sz)ele[i][i]=x;  
    }  
    Matrix operator + (const Matrix b){  
        Matrix c;  
        _for(i,0,sz)  
            _for(j,0,sz)  
                c.ele[i][j]=(ele[i][j]+b.ele[i][j])%p;  
        return c;  
    }  
    Matrix operator * (const int b){  
        Matrix c;  
        _for(i,0,sz)  
            _for(j,0,sz)  
                c.ele[i][j]=1LL*ele[i][j]*b%p;  
        return c;  
    }  
    void operator *= (const Matrix b){  
        Matrix c=*this;  
        _for(i,0,sz)  
            _for(j,0,sz){  
                ele[i][j]=0;  
                _for(k,0,sz)  
                    ele[i][j]=(ele[i][j]+1LL*c.ele[i][k]*b.ele[k][j])%p;  
            }  
    }  
};  
Matrix quick_pow(const Matrix &a,int k){  
    Matrix Ans(1),Base;  
    Base=a;  
    while(k){  
        if(k&1)Ans*=Base;  
        k>>=1;  
        Base*=Base;  
    }  
}
```

```

    }
    return Ans;
}
struct complex{
    long double x,y;
    complex(long double x=0.0,long double y=0.0):x(x),y(y){}
    complex operator + (const complex &b){
        return complex(x+b.x,y+b.y);
    }
    complex operator - (const complex &b){
        return complex(x-b.x,y-b.y);
    }
    complex operator * (const complex &b){
        return complex(x*b.x-y*b.y,x*b.y+y*b.x);
    }
};
int rev[1<<2];
int build(int k){
    int n,pos=0;
    while((1<<pos)<=k)pos++;
    n=1<<pos;
    _for(i,0,n)rev[i]=(rev[i>>1]>>1)|((i&1)<<(pos-1));
    return n;
}
void FFT(complex *f,int n,int type){
    _for(i,0,n)if(i<rev[i])
        swap(f[i],f[rev[i]]);
    complex t1,t2;
    for(int i=1;i<n;i<=1){
        complex w(cos(pi/i),type*sin(pi/i));
        for(int j=0;j<n;j+=(i<<1)){
            complex cur(1.0,0.0);
            _for(k,j,j+i){
                t1=f[k],t2=cur*f[k+i];
                f[k]=t1+t2,f[k+i]=t1-t2;
                cur=cur*w;
            }
        }
    }
    if(type==-1)_for(i,0,n)
        f[i].x/=n,f[i].y/=n;
}
void FFT2(complex *f1,complex *f2,int n){
    FFT(f1,n,1);
    f2[0].x=f1[0].x,f2[0].y=-f1[0].y;
    _for(i,1,n)
        f2[i].x=f1[n-i].x,f2[i].y=-f1[n-i].y;
    complex t1,t2;
    _for(i,0,n){
        t1=f1[i],t2=f2[i];
        f1[i]=complex((t1.x+t2.x)*0.5,(t1.y+t2.y)*0.5);
    }
}

```

```
        f2[i]=complex((t1.y-t2.y)*0.5,(t2.x-t1.x)*0.5);
    }
}
void MTT(int *f,int n1,int *g,int n2,int *ans,int mod){
    static complex
    f1[MAXN<<2],f2[MAXN<<2],g1[MAXN<<2],g2[MAXN<<2],temp[2][MAXN<<2];
    int n=build(n1+n2),m=sqrt(mod)+1;
    _rep(i,0,n1)f1[i].x=f[i]/m,f1[i].y=f[i]%m;
    _rep(i,0,n2)g1[i].x=g[i]/m,g1[i].y=g[i]%m;
    FFT2(f1,f2,n);FFT2(g1,g2,n);
    complex I(0.0,1.0);
    _for(i,0,n){
        temp[0][i]=f1[i]*g1[i]+I*f2[i]*g1[i];
        temp[1][i]=f1[i]*g2[i]+I*f2[i]*g2[i];
    }
    FFT(temp[0],n,-1);FFT(temp[1],n,-1);
    LL a,b,c;
    _rep(i,0,n1+n2){
a=temp[0][i].x+0.5,b=temp[0][i].y+temp[1][i].x+0.5,c=temp[1][i].y+0.5;
        ans[i]=((a%mod*m%mod*m%mod+b%mod*m%mod+c%mod)%mod+mod)%mod;
    }
}
Matrix W;
int a[MAXN<<2],b[MAXN<<2],c[MAXN<<2];
int main()
{
    int k,L,x,y;
    sz=read_int(),k=read_int(),L=read_int(),x=read_int()-1,y=read_int()-1,p=read_int();
    get_G(k);
    _for(i,0,sz)
    _for(j,0,sz)
    W.ele[i][j]=read_int();
    _for(i,0,k){
        Matrix temp=quick_pow(W*gw[i]+Matrix(1),L);
        a[i]=1LL*temp.ele[x][y]*gw[1LL*i*(i-1)/2%k]%p;
    }
    _for(i,0,2*k-1)b[i]=gw[k-1LL*i*(i-1)/2%k]%p;
    reverse(a,a+k);
    MTT(a,k-1,b,2*k-2,c,p);
    int div=quick_pow(k,p-2);
    _for(i,0,k)
    enter(1LL*div*gw[1LL*i*(i-1)/2%k]%p*c[k-1+i]%p);
    return 0;
}
```

## 常系数齐次线性递推

## 算法简介

给定  $a_n = f_{1a_{n-1}} + f_{2a_{n-2}} + \dots + f_{ka_{n-k}}$  以及  $a_0, a_1, \dots, a_{k-1}$  询问  $a_n$  的值。时间复杂度  $O(k \log k \log n)$

## 算法实现

### 洛谷p4723

考虑求斐波那契数列过程  $f_5 = f_4 + f_3 = 2f_3 + f_2 = 3f_2 + 2f_1 = 5f_1 + 3f_0$

从多项式角度上考虑该过程  $f_n = f_{n-1} + f_{n-2}$  的特征多项式为  $x^2 - x - 1$  并且有

$$\begin{aligned} & \begin{pmatrix} 0x^0 & +0x^1 & +0x^2 & +0x^3 & +0x^4 & +1x^5 \\ \equiv 0x^0 & +0x^1 & +0x^2 & +1x^3 & +1x^4 & +0x^5 \\ \equiv 0x^0 & +0x^1 & +1x^2 & +2x^3 & +0x^4 & +0x^5 \\ \equiv 0x^0 & +3x^1 & +2x^2 & +0x^3 & +0x^4 & +0x^5 \\ \equiv 3x^0 & +5x^1 & +0x^2 & +0x^3 & +0x^4 & +0x^5 \end{pmatrix} \pmod{x^2 - x - 1} \end{aligned}$$

于是  $f_5 = \left(x^5 \bmod \{x^2 - x - 1\}\right) \cdot (f_0, f_1) = (3, 5) \cdot (f_0, f_1)$

以此类推，对  $a_n = f_{1a_{n-1}} + f_{2a_{n-2}} + \dots + f_{ka_{n-k}}$  其特征多项式为  $x^k - f_1 x^{k-1} - f_2 x^{k-2} - \dots - f_k$

于是  $a_n = \left(x^n \bmod \{x^k - f_1 x^{k-1} - f_2 x^{k-2} - \dots - f_k\}\right) \cdot (a_0, a_1, \dots, a_{k-1})$

其中  $\left(x^n \bmod \{x^k - f_1 x^{k-1} - f_2 x^{k-2} - \dots - f_k\}\right)$  可以通过快速幂与多项式带余除法  $O(k \log k \log n)$  计算。

发现每次带余除法计算时  $g(x)$  都是固定的，可以考虑先计算出  $\frac{1}{g^R(x)}$  减小常数。

```
const int MAXN=64005,Mod=998244353;
int quick_pow(int a,int b){
    int ans=1;
    while(b){
        if(b&1)
            ans=1LL*ans*a%Mod;
        a=1LL*a*a%Mod;
        b>>=1;
    }
    return ans;
}
namespace Poly{
    const int G=3;
    int rev[MAXN<<2],Wn[30][2];
    void init(){
        int m=Mod-1,lg2=0;
        while(m%2==0)m>>=1,lg2++;
        Wn[lg2][1]=quick_pow(G,m);
    }
}
```

```
Wn[lg2][0]=quick_pow(Wn[lg2][1],Mod-2);
while(lg2){
    m<=<=1,lg2--;
    Wn[lg2][0]=1LL*Wn[lg2+1][0]*Wn[lg2+1][0]%Mod;
    Wn[lg2][1]=1LL*Wn[lg2+1][1]*Wn[lg2+1][1]%Mod;
}
}
int build(int k){
    int n,pos=0;
    while((1<<pos)<=k)pos++;
    n=1<<pos;
    _for(i,0,n)rev[i]=(rev[i>>1]>>1)|((i&1)<<(pos-1));
    return n;
}
void NTT(int *f,int n,bool type){
    _for(i,0,n)if(i<rev[i])
        swap(f[i],f[rev[i]]);
    int t1,t2;
    for(int i=1,lg2=0;i<n;i<=<=1,lg2++){
        int w=Wn[lg2+1][type];
        for(int j=0;j<n;j+=(i<<1)){
            int cur=1;
            _for(k,j,j+i){
                t1=f[k],t2=1LL*cur*f[k+i]%Mod;
                f[k]=(t1+t2)%Mod,f[k+i]=(t1-t2)%Mod;
                cur=1LL*cur*w%Mod;
            }
        }
    }
    if(!type){
        int div=quick_pow(n,Mod-2);
        _for(i,0,n)f[i]=(1LL*f[i]*div%Mod+Mod)%Mod;
    }
}
void Mul(int *f,int _n,int *g,int _m,int xmod=0){
    int n=build(_n+_m-2);
    _for(i,_n,n)f[i]=0;_for(i,_m,n)g[i]=0;
    NTT(f,n,true);NTT(g,n,true);
    _for(i,0,n)f[i]=1LL*f[i]*g[i]%Mod;
    NTT(f,n,false);
    if(xmod)_for(i,xmod,n)f[i]=0;
}
void Inv(const int *f,int *g,int _n){
    static int temp[MAXN<<2];
    if(_n==1)return g[0]=quick_pow(f[0],Mod-2),void();
    Inv(f,g,(_n+1)>>1);
    int n=build((_n-1)<<1);
    _for(i,(_n+1)>>1,n)g[i]=0;
    _for(i,0,_n)temp[i]=f[i];_for(i,_n,n)temp[i]=0;
    NTT(g,n,true);NTT(temp,n,true);
}
```

```

    _for(i,0,n)g[i]=(2-1LL*temp[i]*g[i]%Mod)*g[i]%Mod;
    NTT(g,n,false);
    _for(i,_n,n)g[i]=0;
}
void Div(int *f,int _n,const int *g,int _m,int *gR){
    static int temp[MAXN<<2],q[MAXN<<2],r[MAXN<<2];
    _rep(i,0,_n-_m)q[i]=gR[i];
    _rep(i,0,_n)temp[i]=f[_n-i];
    Mul(q,_n-_m+1,temp,_n+1,_n-_m+1);
    for(int i=0,j=_n-_m;i<j;i++,j--)swap(q[i],q[j]);
    int __m=min(_n-_m+1,_m);
    _for(i,0,__m)r[i]=g[i];_for(i,0,__m)temp[i]=q[i];
    Mul(r,__m,temp,__m,__m);
    _for(i,0,__m)f[i]=(f[i]+Mod-r[i])%Mod;
}
void Pow_2(int *f,int _n,const int *g,int _m){
    static int temp1[MAXN<<2],temp2[MAXN<<2];
    _rep(i,0,__m)temp1[i]=g[_m-i];
    Inv(temp1,temp2,__m-1);
    mem(temp1,0);
    temp1[0]=1;
    while(_n){
        int n=build(__m+__m-2);
        _for(i,__m,n)f[i]=0;
        NTT(f,n,true);
        if(_n&1){
            _for(i,__m,n)temp1[i]=0;
            NTT(temp1,n,true);
            _for(i,0,n)temp1[i]=1LL*f[i]*temp1[i]%Mod;
            NTT(temp1,n,false);
        }
        _for(i,0,n)f[i]=1LL*f[i]*f[i]%Mod;
        NTT(f,n,false);
        if(_n&1)Div(temp1,__m+__m-2,g,__m,temp2);
        Div(f,__m+__m-2,g,__m,temp2);
        _n>>=1;
    }
    mem(f,0);
    _for(i,0,__m)f[i]=temp1[i];
}
}
int f[MAXN<<2],g[MAXN<<2],a[MAXN];
int main()
{
    Poly::init();
    int n=read_int(),k=read_int();
    g[k]=1;
    for(int i=k-1;i>=0;i--)g[i]=-read_int();
    _for(i,0,k)a[i]=read_int();
    f[1]=1;
    Poly::Pow_2(f,n,g,k);
}

```

```
int ans=0;
_for(i,0,k)ans=(ans+1LL*f[i]*a[i])%Mod;
enter((ans+Mod)%Mod);
return 0;
}
```

## 多项式多点求值

### 算法简介

给定一个  $n$  次多项式  $f(x)$  求  $f(a_i)$  ( $1 \leq i \leq m$ ) 时间复杂度  $O(n \log n + m \log^2 m)$  空间复杂度  $O(m \log m)$

### 算法实现

#### 洛谷p5050

设  $f(x) = c_0 + c_1x + c_2x^2 + \dots + c_nx^n$  于是有

$$\begin{aligned} f(x) - f(x_0) &\equiv c_0 - c_0 + c_1(x - x_0) + c_2(x^2 - x_0^2) + \dots \\ &+ c_n(x^n - x_0^n) \equiv c_1(x - x_0) + c_2(x - x_0)(x + x_0) + \dots + c_n(x - x_0)(x^{n-1} + x^{n-2}x_0 + \dots + x_0^{n-1}) \equiv 0 \pmod{x - x_0} \end{aligned}$$

即  $f(x_0) \equiv f(x) \pmod{x - x_0}$  于是如果能快速计算  $f(x) \bmod x - x_0$  就可以得到  $f(x_0)$

记  $f_{\{l,r\}}(x) \equiv f(x) \pmod{\prod_{i=l}^r (x - a_i)}$  于是有  $f_{\{l,r\}}(x) \equiv f_{\{l,m\}}(x) \pmod{\prod_{i=l}^m (x - a_i)}$ ,  $f_{\{l,r\}}(x) \equiv f_{\{m+1,r\}}(x) \pmod{\prod_{i=m+1}^r (x - a_i)}$

考虑  $O(m \log^2 m)$  自底向上建立线段树，每个节点存储  $\prod_{i=l}^r (x - a_i)$  的多项式。

然后  $O(m \log^2 m)$  自顶向下进行带余除法，跑到叶子节点就可以得到每个  $f(a_i)$  的值。

注意根节点需要提前进行带余除法，同时该算法常数较大，考虑常数优化。

不妨令  $r-l$  小于某个固定值  $L$  后采用  $O(L^2)$  暴力秦九韶展开，秦九韶复杂度为  $O(\frac{mL^2}{L}) = O(mL)$

于是总时间复杂度变为  $O(n \log n + m \log m (\log m - \log L) + mL)$  经检验  $m=64000$  时取  $L=640$  效果较佳。

```
const int MAXN=64005,MINL=640,Mod=998244353;
int quick_pow(int a,int b){
    int ans=1;
    while(b){
        if(b&1)
```

```

    ans=1LL*ans*a%Mod;
    a=1LL*a*a%Mod;
    b>>=1;
}
return ans;
}
namespace Poly{
const int G=3;
int rev[1<<2],Pool[1<<3],*Wn[30];
void init(){
    int lg2=0,*pos=Pool,n,w;
    while((1<<lg2)<MAXN*2)lg2++;
    n=1<<lg2,w=quick_pow(G,(Mod-1)/(1<<lg2));
    while(~lg2){
        Wn[lg2]=pos,pos+=n;
        Wn[lg2][0]=1;
        _for(i,1,n)Wn[lg2][i]=1LL*Wn[lg2][i-1]*w%Mod;
        w=1LL*w*w%Mod;
        lg2--;n>>=1;
    }
}
int build(int k){
    int n,pos=0;
    while((1<<pos)<=k)pos++;
    n=1<<pos;
    _for(i,0,n)rev[i]=(rev[i>>1]>>1)|((i&1)<<(pos-1));
    return n;
}
void NTT(int *f,int n,bool type){
    _for(i,0,n)if(i<rev[i])
        swap(f[i],f[rev[i]]);
    int t1,t2;
    for(int i=1,lg2=1;i<n;i<=1,lg2++){
        for(int j=0;j<n;j+=(i<<1)){
            _for(k,j,j+i){
                t1=f[k],t2=1LL*Wn[lg2][k-j]*f[k+i]%Mod;
                f[k]=(t1+t2)%Mod,f[k+i]=(t1-t2)%Mod;
            }
        }
    }
    if(!type){
        reverse(f+1,f+n);
        int div=quick_pow(n,Mod-2);
        _for(i,0,n)f[i]=(1LL*f[i]*div%Mod+Mod)%Mod;
    }
}
void Mul(int *f,int _n,int *g,int _m,int xmod=0){
    int n=build(_n+_m-2);
    _for(i,_n,n)f[i]=0;_for(i,_m,n)g[i]=0;
    NTT(f,n,true);NTT(g,n,true);
    _for(i,0,n)f[i]=1LL*f[i]*g[i]%Mod;
}

```

```
NTT(f,n,false);
if(xmod)_for(i,xmod,n)f[i]=0;
}
void Mul2(const int *f,int _n,const int *g,int _m,int *h){
static int temp1[MAXN<<2],temp2[MAXN<<2];
int n=build(_n+_m-2);
memcpy(temp1,f,sizeof(int)*_n);memcpy(temp2,g,sizeof(int)*_m);
_for(i,_n,n)temp1[i]=0;_for(i,_m,n)temp2[i]=0;
NTT(temp1,n,true);NTT(temp2,n,true);
_for(i,0,n)temp1[i]=1LL*temp1[i]*temp2[i]%Mod;
NTT(temp1,n,false);
n=_n+_m-1;
_for(i,0,n)h[i]=temp1[i];
}
void Inv(const int *f,int *g,int _n){
static int temp[MAXN<<2];
if(_n==1)return g[0]=quick_pow(f[0],Mod-2),void();
Inv(f,g,(_n+1)>>1);
int n=build(( _n-1)<<1);
_for(i,(_n+1)>>1,n)g[i]=0;
_for(i,0,_n)temp[i]=f[i];_for(i,_n,n)temp[i]=0;
NTT(g,n,true);NTT(temp,n,true);
_for(i,0,n)g[i]=(2-1LL*temp[i]*g[i]%Mod)*g[i]%Mod;
NTT(g,n,false);
_for(i,_n,n)g[i]=0;
}
void Div(const int *f,int _n,const int *g,int _m,int *r){
static int temp1[MAXN<<2],temp2[MAXN<<2];
if(_n<_m){
_rep(i,0,_n)r[i]=f[i];
return;
}
_rep(i,0,_m)temp1[i]=g[_m-i];
Inv(temp1,temp2,_n-_m+1);
_rep(i,0,_n)temp1[i]=f[_n-i];
Mul(temp2,_n-_m+1,temp1,_n+1,_n-_m+1);
for(int i=0,j=_n-_m;i<j;i++,j--)swap(temp2[i],temp2[j]);
int __m=min(_n-_m+1,_m);
_for(i,0,__m)temp1[i]=g[i];
Mul(temp1,__m,temp2,__m);
_for(i,0,__m)r[i]=(f[i]+Mod-temp1[i])%Mod;
}
}
int a[MAXN],b[MAXN],c[MAXN],pool[MAXN*40],*pos=pool,*g[MAXN<<2];
void build(int k,int L,int R){
g[k]=pos,pos+=R-L+2;
int M=L+R>>1;
if(L==R)return g[k][0]=Mod-a[M],g[k][1]=1,void();
build(k<<1,L,M);build(k<<1|1,M+1,R);
Poly::Mul2(g[k<<1],M-L+2,g[k<<1|1],R-M+1,g[k]);
}
```

```

}
void query(int k,int L,int R,int *f){
    if(R-L<MINL){
        _rep(i,L,R){
            b[i]=0;
            for(int j=R-L;~j;j--)
                b[i]=(1LL*b[i]*a[i]+f[j])%Mod;
        }
        return;
    }
    int *temp=pos,M=L+R>>1;pos+=R-L+1;
    Poly::Div(f,R-L,g[k<<1],M-L+1,temp);
    query(k<<1,L,M,temp);
    Poly::Div(f,R-L,g[k<<1|1],R-M,temp);
    query(k<<1|1,M+1,R,temp);
}
int main()
{
    Poly::init();
    int n=read_int(),m=read_int();
    _rep(i,0,n)c[i]=read_int();
    _for(i,0,m)a[i]=read_int();
    build(1,0,m-1);
    int *f=pos;pos+=m;
    Poly::Div(c,n,g[1],m,f);
    query(1,0,m-1,f);
    _for(i,0,m)enter(b[i]);
    return 0;
}

```

## 多项式快速插值

From:  
<https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link:  
[https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal\\_string:jxm2001:%E5%A4%9A%E9%A1%B9%E5%BC%8F\\_4&rev=1598330660](https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:%E5%A4%9A%E9%A1%B9%E5%BC%8F_4&rev=1598330660)

Last update: 2020/08/25 12:44