云某人的数学

快速幂

mod 自变量

```
int qpow(int a,int b,int p) // 快速幂
{
    int z = 1;
    a %= p;
    while(b)
    {
        if(b&1) z = (z*a)%p;
        a = (a*a)%p;
        b >>= 1;
    }
    return z;
}
```

mod 宏定义

```
int qpow(int a,int b)
{
    int z = 1;
    a %= M;
    while(b)
    {
        if(b&1) z = (z*a)%M;
        a = (a*a)%M;
        b >>= 1;
    }
    return z;
}
```

组合数

n, m 较小

用二维数组的杨辉三角模拟
$C[i][j] = C_i^j$

```
c[0][0] = 1;
for(int i=0;i<=k;++i)
{
    for(int j=0;j<=i;++j)   c[i][j] = (c[i-1][j-1] + c[i-1][j])%M;
}
```

n, m 较大

$$C_n^m = \frac{n!}{(n-m)! \cdot m!}$$
逆元分数即可。

```
const int N = 2e5+10;
const int M = 1e9+7;

times[0] = 1;// 当 n = m 时，t[n-m] 应该为 1 而不是 0。
for(int i=1;i<=N;++i)   times[i] = (times[i-1]*i)%M;

int times[N];
int qpow(int a,int b) // 快速幂
{
    int z = 1;
    while(b)
    {
        if(b&1) z = (z*a)%M;
        a = (a*a)%M;
        b >>= 1;
    }
    return z;
}
```

```
int inv(int x) // 逆元
{
    return qpow(x,M-2);
}

int C(int x,int y) // 组合数
{
    return (times[x]*inv(times[x-y]*times[y]%M)%M);
}
```

线性筛

```
int primes[N], cnt;     // primes[]存储所有素数
bitset<N> vis;         // st[x]存储x是否被筛掉

void get_primes(int n)
{
    for (int i = 2; i <= n; i ++ )
    {
        if (!vis[i]) primes[cnt ++ ] = i;
        for (int j = 0; primes[j] <= n / i; j ++ )
        {
            vis[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}
```

fft

快速傅里叶变换

fft 求解多项式乘法的 n 个结果，$ifft$（逆变换）求解系数，具体使用线代逆矩阵性质。
这样可以求解多项式系数。

高精度乘法

保证存储系数的数组长度 $\geq$ 被计算的两个数长度和的两倍

高精度 $\times$ 高精度理解为两个多项式相乘，乘出的结果多项式的系数就是乘法计算的结果，因为代入对应进制及是对应的数。

code

复数

```
struct cpx
{
    double x,y;
    cpx operator + (const cpx i)    {return {x+i.x,y+i.y};}
    cpx operator - (const cpx i)    {return {x-i.x,y-i.y};}
    cpx operator * (const cpx i)    {return {x*i.x-y*i.y,x*i.y+y*i.x};}
};
```

归并

```
void fft(cpx a[],int L,int op)
{
    if(L == 1)  return;
    cpx a1[L>>1], a2[L>>1];
    for(int i=0;i<(L>>1);++i)   a1[i] = a[i<<1], a2[i] = a[i<<1|1];
    fft(a1,L>>1,op), fft(a2,L>>1,op);
    cpx wk = {cos(2*pi/L),sin(2*pi/L)*op}, w = {1,0};
    for(int i=0;i<(L>>1);++i,w=w*wk)
    {
        a[i] = a1[i] + a2[i]*w;
        a[i+(L>>1)] = a1[i] - a2[i]*w;
    }
}
```

迭代

```
int r[N];
cpx a[N],b[N];
void fft(cpx a[],int L,int op)
{
    // 反转变换
    for(int i=0;i<L;++i)    r[i] = r[i>>1]/2 + (i&1 ? L>>1 : 0);
    for(int i=0;i<L;++i)
    {
        if(i < r[i])    swap(a[i],a[r[i]]);
    }
    for(int len=2;len<=L;len<<=1)
    {
        cpx wk = {cos(2*pi/len),sin(2*pi/len)*op};
        for(int i=0;i<L;i+=len)
        {
            cpx w = {1,0};
            for(int j=0;j<(len>>1);++j,w=w*wk)
            {
                cpx x = a[i+j], y = a[i+j+(len>>1)]*w;
                a[i+j] = x+y, a[i+j+(len>>1)] = x-y;
            }
        }
    }
}
```

ntt

> 叠甲：因为本人不是数论手，记录的只为个人理解，可能不太准确。

因为 fft 涉及复数，计算用到了浮点数，容易产生浮点误差，所以我们思考能否使用整数来实现单位圆复数的性质。

利用原根的性质，也可以实现周期性

模数 p 需要是 $q \times 2^k + 1$ 的素数

原根	模数 p	分解 p	最大长度
3	49762049	$7 \times 2^{26} + 1$	2^{26}
3	998244353	$119 \times 2^{23} + 1$	2^{26}
3	2291701377	$17 \times 2^{27} + 1$	2^{27}

求逆则直接乘逆元

归并

```
const int P = 998244353;
const int G = 3;
const int inv_G = 332748118;// inv(g)

void ntt(int a[],int L,int op)
{
    if(L == 1)  return;
    int a1[L>>1], a2[L>>1];
    for(int i=0;i<(L>>1);++i)   a1[i] = a[i<<1], a2[i] = a[i<<1|1];
    ntt(a1,L>>1,op), ntt(a2,L>>1,op);
    int gk = qpow(op == 1 ? G : inv_G,(P-1)/L), g = 1;
    for(int i=0;i<(L>>1);++i,g=g*gk%P)
    {
        a[i] = (a1[i] + a2[i]*g%P)%P;
        a[i+(L>>1)] = (a1[i] - a2[i]*g%P+P)%P;
    }
}
```

迭代

和fft一模一样

```
void ntt(vector<int> &a,int L,int op)
{
    vector<int> r(L);
    for(int i=0;i<L;++i)    r[i] =  r[(i>>1)]/2+(i&1 ? L>>1 : 0);
    for(int i=0;i<L;++i)
    {
        if(i < r[i])    swap(a[i],a[r[i]]);
    }
    for(int i=2;i<=L;i<<=1)
    {
        int gk = qpow(op==1 ? G : inv_G,(P-1)/i);
        for(int j=0;j<L;j+=i)
        {
            int g = 1;
            for(int k=j;k<j+(i>>1);++k,g=g*gk%P)
            {
                int x = a[k], y = a[k+(i>>1)]*g%P;
                a[k] = (x+y)%P, a[k+(i>>1)] = (x-y+P)%P;
            }
        }
```

```
    }
}
```

高精度

乘法

FFT

需要给被乘的两个数设置为复数

```
struct cpx
{
    double x,y;
    cpx operator + (const cpx i)    {return {x+i.x,y+i.y};}
    cpx operator - (const cpx i)    {return {x-i.x,y-i.y};}
    cpx operator * (const cpx i)    {return {x*i.x-y*i.y,x*i.y+y*i.x};}
};

vector<cpx> a,b;

void fft(vector<cpx> &a,int L,int op)
{
    // 反转变换
    vector<int> r(L);
    for(int i=0;i<L;++i)    r[i] = r[i>>1]/2 + (i&1 ? L>>1 : 0);
    for(int i=0;i<L;++i)
    {
        if(i < r[i])    swap(a[i],a[r[i]]);
    }
    for(int i=2;i<=L;i<<=1)
    {
        cpx wk = {cos(2*pi/i),sin(2*pi/i)*op};
        for(int j=0;j<L;j+=i)
        {
            cpx w = {1,0};
            for(int k=j;k<(j+(i>>1));++k,w=w*wk)
            {
                cpx x = a[k], y = a[k+(i>>1)]*w;
                a[k] = x+y, a[k+(i>>1)] = x-y;
            }
        }
    }
    if(op == -1)
    {
        for(int i=0;i<L;++i)    a[i].x /= L, a[i].y /= L;
    }
}

vector<int> mul(vector<cpx> &a,vector<cpx> &b)
{
    vector<int> res;
    int n = a.size()+b.size(), L = 1;
    while(n >= L)    L <<= 1;
    a.resize(L), b.resize(L);
```

```
    fft(a,L,1), fft(b,L,1);
    for(int i=0;i<L;++i)    a[i] = a[i]*b[i];
    fft(a,L,-1);
    int t = 0;
    for(int i=0;i<n-1||t;++i)
    {
        t += (a[i].x+0.5);
        res.push_back(t%10);
        t /= 10;
    }
    return res;
}
```

NTT

```
const int P = 998244353;
const int G = 3;
const int inv_G = 332748118;// inv(g)

int qpow(int a,int b)
{
    int z = 1;
    a %= P;
    while(b)
    {
        if(b&1) z = (z*a)%P;
        a = (a*a)%P;
        b >>= 1;
    }
    return z;
}

void ntt(vector<int> &a,int L,int op)
{
    vector<int> r(L);
    for(int i=0;i<L;++i)    r[i] =  r[(i>>1)]/2+(i&1 ? L>>1 : 0);
    for(int i=0;i<L;++i)
    {
        if(i < r[i])    swap(a[i],a[r[i]]);
    }
    for(int i=2;i<=L;i<<=1)
    {
        int gk = qpow(op==1 ? G : inv_G,(P-1)/i);
        for(int j=0;j<L;j+=i)
        {
            int g = 1;
            for(int k=j;k<j+(i>>1);++k,g=g*gk%P)
            {
                int x = a[k], y = a[k+(i>>1)]*g%P;
                a[k] = (x+y)%P, a[k+(i>>1)] = (x-y+P)%P;
            }
        }
    }
    if(op == -1)
    {
```

```
        int inv_L = qpow(L,P-2);
        for(int i=0;i<L;++i)    a[i] = a[i]*inv_L%P;
    }
}

vector<int> mul(vector<int> &a,vector<int> &b)
{
    vector<int> res;
    int n = a.size()+b.size(), L = 1;
    while(n >= L)   L <<= 1;
    a.resize(L), b.resize(L);
    ntt(a,L,1), ntt(b,L,1);
    for(int i=0;i<L;++i)    a[i] = a[i]*b[i]%P;
    ntt(a,L,-1);
    int t = 0;
    for(int i=0;i<n-1||t;++i)
    {
        t += a[i];
        res.push_back(t%10);
        t /= 10;
    }
    return res;
}
```