

多项式 3

分治 FFT

算法简介

$O(n \log^2 n)$ 时间解决一些难以直接使用 FFT 解决的问题。

算法例题

[洛谷p4721](#)

题意

给定 $g_0, g_1 \cdots g_{n-2}$

已知 $f_0=1, f_{i+1}=\sum_{j=0}^i f_j g_{i-j}$ 求 $f_0, f_1 \cdots f_{n-1}$

题解

发现转移过程可以用 CDQ 分治优化，区间 $[\text{lef}, \text{mid}]$ 对区间 $[\text{mid}, \text{rig}]$ 的贡献为

$$f_{i+1} \text{ gets } \sum_{j=\text{lef}}^{\text{mid}} f_j g_{i-j}$$

套用 NTT 可以 $O(n \log n)$ 求出 $\sum_{j=\text{lef}}^{\text{mid}} f_j g_{i-j}, (\text{mid} \leq i < \text{rig})$ 于是总时间复杂度为 $O(n \log^2 n)$

```
const int MAXN=1e5+5,Mod=998244353,G=3,Inv_G=332748118;
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;
}
int rev[MAXN<<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 NTT(int *f,int n,int type){
    _for(i,0,n)if(i<rev[i])
        swap(f[i],f[rev[i]]);
    int t1,t2;
    for(int i=1;i<n;i<=<1){
        int w=quick_pow(type==1?G:Inv_G,(Mod-1)/(i<=<1));
        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==-1){
        int div=quick_pow(n,Mod-2);
        _for(i,0,n)
            f[i]=(1LL*f[i]*div%Mod+Mod)%Mod;
    }
}
int f[MAXN],g[MAXN],t1[MAXN<<2],t2[MAXN<<2];
void solve(int lef,int rig){
    if(lef==rig)return;
    int mid=lef+rig>>1;
    solve(lef,mid);
    int n1=mid-lef,n2=rig-lef-1,n=build(n1+n2);
    _rep(i,0,n1)t1[i]=f[i+lef];_for(i,n1+1,n)t1[i]=0;
    _rep(i,0,n2)t2[i]=g[i];_for(i,n2+1,n)t2[i]=0;
    NTT(t1,n,1);NTT(t2,n,1);
    _for(i,0,n)t1[i]=1LL*t1[i]*t2[i]%Mod;
    NTT(t1,n,-1);
    _for(i,mid,rig)f[i+1]=(f[i+1]+t1[i-lef])%Mod;
    solve(mid+1,rig);
}
int main()
{
    int n=read_int();
    _for(i,0,n-1)
        g[i]=read_int();
    f[0]=1;
    solve(0,n-1);
    _for(i,0,n)
        space(f[i]);
    return 0;
}
```

多项式求逆

算法简介

给定 $f(x)$ 求 $f(x)f^{-1}(x) \equiv 1 \pmod{x^n}$ 时间复杂度 $O(n \log n)$

算法实现

假设已知 $f(x)f_0^{-1}(x) \equiv 1 \pmod{x^{\lceil \frac{n}{2} \rceil}}$

由于 $f(x)f^{-1}(x) \equiv 1 \pmod{x^n}$ 显然有 $f(x)f^{-1}(x) \equiv 1 \pmod{x^{\lceil \frac{n}{2} \rceil}}$

于是 $f^{-1}(x) - f_0^{-1}(x) \equiv 0 \pmod{x^{\lceil \frac{n}{2} \rceil}}$

两倍同时平方，有 $f^{-2}(x) - 2f^{-1}(x)f_0^{-1}(x) + f_0^{-2}(x) \equiv 0 \pmod{x^n}$

两边同时乘以 $f(x)$ 有 $f^{-1}(x) \equiv f_0^{-1}(x)(2 - f(x)f_0^{-1}(x)) \pmod{x^n}$

现在考虑逆元存在条件，发现只要 $[x^0]f(x)$ 的逆元存在，就可以递推出 $f(x)$ 的逆元。

于是 $f^{-1}(x)$ 存在等价于 $\left([x^0]f(x)\right)^{-1}$ 存在。

时间复杂度有 $T(n) = T\left(\frac{n}{2}\right) + O(n \log n)$ 于是 $T(n) = O(n \log n)$

递归版与递推版效率相差不大。

```
// 递归版
int temp[MAXN<<2];
void ployinv(int *f,int *g,int n){
    if(n==1)
        return g[0]=quick_pow(f[0],Mod-2),void();
    ployinv(f,g,(n+1)>>1);
    int m=build(n<<1);
    _for(i,0,n)temp[i]=f[i];_for(i,n,m)temp[i]=0;
    NTT(temp,m,1);NTT(g,m,1);
    _for(i,0,m)g[i]=(2-1LL*temp[i]*g[i]%Mod)*g[i]%Mod;
    NTT(g,m,-1);
    _for(i,n,m)g[i]=0;
}

// 递推版
int temp[MAXN<<2];
void ployinv(int *f,int *g,int n){
    g[0]=quick_pow(f[0],Mod-2);
    int n1=2,n2=4,pos=2;
    while((n1>>1)<n){
        _for(i,0,n2)rev[i]=(rev[i>>1]>>1)|((i&1)<<(pos-1));
        _for(i,0,n1)temp[i]=f[i];_for(i,n1,n2)temp[i]=0;
        NTT(temp,n2,1);NTT(g,n2,1);
        _for(i,0,n2)g[i]=(2-1LL*temp[i]*g[i]%Mod)*g[i]%Mod;
    }
}
```

```
NTT(g, n2, -1);  
_for(i, n1, n2) g[i] = 0;  
n1 <<= 1, n2 <<= 1, pos++;  
}  
n1 >>= 1;  
_for(i, n, n1) g[i] = 0;  
}
```

多项式开方

算法简介

给定 $g(x)$ 求 $f^2(x) \equiv g(x) \pmod{x^n}$ 时间复杂度 $O(n \log n)$

算法实现

假设已知 $f_0^2(x) \equiv g(x) \pmod{x^{\lceil \frac{n}{2} \rceil}}$

两边平方，有 $\left(f_0^2(x) - g(x)\right) \equiv 0 \pmod{x^n}$

两边加上 $4f_0^2(x)g(x)$ 有 $\left(f_0^2(x) + g(x)\right)^2 \equiv 4f_0^2(x)g(x) \pmod{x^n}$

两边除以 $4f_0^2(x)$ 有 $\left(\frac{f_0^2(x) + g(x)}{2f_0^2(x)}\right)^2 \equiv g(x) \pmod{x^n}$

于是有 $f(x) \equiv \frac{f_0^2(x) + g(x)}{2f_0^2(x)} \equiv \frac{f_0(x) + f_0^{-1}(x)g(x)}{2} \pmod{x^n}$

现在考虑 $f(x)$ 存在条件，发现只要 $([x^0]f(x))^2 \equiv [x^0]g(x) \pmod{p}$ 有解即可。

考虑 BSGS 求出 $[x^0]g(x)$ 对应原根的幂次，即可得到 $[x^0]f(x)$

```
HASH_Table<int, int> H;  
int bsgs(int a, int b) {  
    H.clear();  
    int m = sqrt(Mod) + 1, t = b, base;  
    for(int i = 1; i <= m; i++) {  
        t = 1LL * t * a % Mod;  
        H.insert(t, i);  
    }  
    t = 1, base = quick_pow(a, m);  
    for(int i = 1; i <= m; i++) {  
        t = 1LL * t * base % Mod;  
        if(H.find(t) != -1) return m * i - H.find(t);  
    }  
    return -1;  
}  
int temp[MAXN << 2], inv_f[MAXN << 2];
```

```

void ploysqrt(int *f,int *g,int n){
    f[0]=quick_pow(3,bsgs(3,g[0])/2);
    int n1=2,n2=4,pos=2,inv2=quick_pow(2,Mod-2);
    while((n1>>1)<n){
        _for(i,0,n2) rev[i]=(rev[i>>1]>>1)|((i&1)<<(pos-1));
        _for(i,0,n2) inv_f[i]=0;
        ployinv(f,inv_f,n1);
        _for(i,0,n1) temp[i]=g[i];_for(i,n1,n2) temp[i]=0;
        NTT(inv_f,n2,1);NTT(temp,n2,1);
        _for(i,0,n2) temp[i]=1LL*temp[i]*inv_f[i]%Mod;
        NTT(temp,n2,-1);
        _for(i,0,n1) f[i]=1LL*(f[i]+temp[i])*inv2%Mod;
        n1<<=1,n2<<=1,pos++;
    }
    n1>>=1;
    _for(i,n,n1) f[i]=0;
}

```

多项式对数函数

算法简介

给定 $f(x)$ 求模 x^n 意义下的 $\ln f(x)$ 时间复杂度 $O(n \log n)$

算法实现

$$\frac{d}{dx}(\ln f(x)) \equiv \frac{f'(x)}{f(x)} \pmod{x^n}$$

$$\ln f(x) - \ln f(0) \equiv \int_0^x \frac{f'(t)}{f(t)} \pmod{x^n}$$

由于一般只考虑 $f(0)=1$ 的情况，同时易知 $\int f'(x) f^{-1}(x)$ 常数项为 0 ，于是有

$$\ln f(x) \equiv \int f'(x) f^{-1}(x) \pmod{x^n}$$

```

int inv_f[MAXN<<2];
void ployln(int *f,int n){
    mem(inv_f,0);
    ployinv(f,inv_f,n);
    int m=build((n-1)<<1);
    _rep(i,1,n) f[i-1]=1LL*f[i]*i%Mod;_for(i,n,m) f[i]=0;
    NTT(f,m,1);NTT(inv_f,m,1);
    _for(i,0,m) f[i]=1LL*f[i]*inv_f[i]%Mod;
    NTT(f,m,-1);
    for(int i=n-1;i>=0;i--) f[i]=1LL*f[i-1]*quick_pow(i,Mod-2)%Mod;
    f[0]=0;
    _for(i,n,m) f[i]=0;
}

```

多项式牛顿迭代法

算法简介

给定多项式 $g(x)$ 求 $f(x)$ 满足 $g(f(x)) \equiv 0 \pmod{x^n}$ 时间复杂度 $O(n \log n)$

算法实现

首先单独求出 $[x^0]g(f(x)) \equiv 0 \pmod{x}$ 假设已知 $g(f_0(x)) \equiv 0 \pmod{x^{\lceil \frac{n}{2} \rceil}}$

将 $g(x)$ 在 $f_0(x)$ 处泰勒展开, 有

$$\sum_{i=0}^{\infty} \frac{g^{(i)}(f_0(x))}{i!} (f(x) - f_0(x))^i \equiv 0 \pmod{x^n}$$

同时有 $x^{\lceil \frac{n}{2} \rceil} \mid (f(x) - f_0(x))$ 于是有 $(f(x) - f_0(x))^i \equiv 0 \pmod{x^n} (i \geq 2)$

$$\sum_{i=0}^{\infty} \frac{g^{(i)}(f_0(x))}{i!} (f(x) - f_0(x))^i \equiv g(f_0(x)) + g'(f_0(x))(f(x) - f_0(x)) \equiv 0 \pmod{x^n}$$

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

准确来说这里把 $f_0(x)$ 当成了变元 y $g'(f_0(x)) = \frac{\partial g}{\partial y}(y, x)$

举个例子 $g(f_0(x)) = g(y, x) = xy + x^2 + x = \frac{x}{f_0(x)} + x^2 + x$ $g'(f_0(x)) = \frac{\partial g}{\partial y}(y, x) = -\frac{x}{f_0^2(x)} = -\frac{x}{f_0^2(x)}$

多项式指数函数

算法简介

给定 $f(x)$ 求模 x^n 意义下的 $\exp f(x)$ 时间复杂度 $O(n \log n)$

算法实现

考虑牛顿迭代法, 设 $F(x) \equiv \exp f(x) \pmod{x^n}$ 于是有 $g(F(x)) \equiv \ln F(x) - f(x) \equiv 0 \pmod{x^n}$

$$F(x) \equiv F_0(x) - \frac{g(F_0(x))}{g'(F_0(x))} \equiv F_0(x) - \frac{\ln F_0(x) - f(x)}{1/F_0(x)} \equiv F_0(x) \left(1 + (f(x) - F_0(x))\right) \pmod{x^n}$$

```
int ln_g[MAXN<<2];
void ployexp(int *f, int *g, int n){
    g[0]=1;
```

```

int n1=2,n2=4,pos=2;
while((n1>>1)<n){
    _for(i,0,n1>>1)ln_g[i]=g[i];_for(i,n1>>1,n2)ln_g[i]=0;
    ployln(ln_g,n1);
    ln_g[0]=(1+f[0]-ln_g[0])%Mod;
    _for(i,1,n1)ln_g[i]=(f[i]-ln_g[i])%Mod;
    _for(i,0,n2)rev[i]=(rev[i>>1]>>1)|((i&1)<<(pos-1));
    NTT(g,n2,1);NTT(ln_g,n2,1);
    _for(i,0,n2)g[i]=1LL*g[i]*ln_g[i]%Mod;
    NTT(g,n2,-1);
    _for(i,n1,n2)g[i]=0;
    n1<<=1,n2<<=1,pos++;
}
n1>>=1;
_for(i,n,n1)g[i]=0;
}

```

多项式指数函数

算法简介

给定 $f(x)$ 求模 x^n 意义下的 $f^k(x)$ 时间复杂度 $O(n \log n)$

算法实现

考虑取对数将幂次运算转化为乘法运算加速算法。而多项式取对数存在 $[x^0]f(x)=1$ 的限制，大多数情况下无法直接套用。

于是考虑选取 $f(x)$ 第一个非零的项，即为 $a_t x^t$ 然后提取出 $a_t x^t$ 得到下式

$$f^k(x) \equiv a_t^k x^{tk} \exp\left(k \ln \frac{f(x)}{a_t x^t}\right) \pmod{x^n}$$

注意到如果 k 为高精度数，需要同时记录 $k \bmod \{p-1\}$ 和 $k \bmod p$ 的结果。

其中计算 a_t^k 需要 $k \bmod \{p-1\}$ 计算 $k \ln \frac{f(x)}{a_t x^t}$ 需要 $k \bmod p$ 同时考虑提前处理 x^{tk} 次数大于 x^n 的情况。

```

int ln_f[MAXN<<2];
void ploypow(int *f,int n,int k1,int k2){
    LL pos=0,posv;
    while(!f[pos]&&pos<n)pos++;
    if(pos==n)return;
    posv=quick_pow(f[pos],Mod-2);
    _for(i,pos,n)ln_f[i-pos]=f[i]*posv%Mod,f[i]=0;
    _for(i,n-pos,n)ln_f[i]=0;
    ployln(ln_f,n);
    _for(i,0,n)ln_f[i]=1LL*ln_f[i]*k1%Mod;
    ployexp(ln_f,f,n);
}

```

```
pos=pos*k2;posv=quick_pow(posv,1LL*k2*(Mod-2)%(Mod-1));  
for(int i=n-1;i>=pos;i--)f[i]=f[i-pos]*posv%Mod;  
pos=min(pos,1LL*n);  
_for(i,0,pos)f[i]=0;  
}
```

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_3&rev=1597150863

Last update: **2020/08/11 21:01**