多项式 3

倍增 FFT

算法简介

主要用于加速可以倍增转移的动态规划，时间复杂度 $O\left(n\log^2 n\right)$。

算法例题

CF755G

题意

将 n 个球排成一排。规定一个组至少包含一个球，至多包含两个相邻的球，且一个球至多属于一个组。

问从这 n 个球中取 k 组有多少方案，答案对 998244353 取模。

题解

设 $\text{dp}(i,j)$ 表示从 i 个球里选 j 组的方案数。

对于第 n 个球，要么不选，要么单独构成一个组，要么与第 $n-1$ 个球构成一个组，于是有状态转移方程

$$\text{dp}(i,j)=\text{dp}(i-1,j)+\text{dp}(i-1,j-1)+\text{dp}(i-1,j-2)$$

同时将 $a+b$ 个球划分为两组 $1\sim a$ 和 $a+1\sim a+b$。根据 $a,a+1$ 是否共同为一组可得状态转移方程

$$\text{dp}(a+b,k)=\sum_{i=0}^k\text{dp}(a,i)\text{dp}(b,k-i)+\sum_{i=0}^{k-1}\text{dp}(a-1,i)\text{dp}(b-1,k-i-1)$$

设 $F_n(x)=\sum_{i=0}^{\infty} \text{dp}(n,i)x^i$。于是有

$$F_n(x)=F_{n-1}(x)+xF_{n-1}(x)+x^2F_{n-2}(x)$$

$$F_{a+b}(x)=F_a(x)F_b(x)+xF_{a-1}(x)F_{b-1}(x)$$

$$F_{2n-2}(x)=F_{n-1}^2(x)+xF_{n-2}^2(x)$$

$$F_{2n-1}(x)=F_n(x)F_{n-1}(x)+xF_{n-1}(x)F_{n-2}(x)$$

$$F_{2n}(x)=F_n^2(x)+xF_{n-1}^2(x)$$

于是可以 $\left(F_{n-2},F_{n-1},F_n\right)\to

\left(F_{n-1},F_n,F_{n+1}\right),\left(F_{2n-2},F_{2n-1},F_{2n}\right)$

于是用类似快速幂的算法可以 $O(k\log n\log k)$ 解决上述问题。

```
const int MAXN=1<<16,Mod=998244353,G=3;
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],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;
    }
}
int f[3][MAXN],temp[5][MAXN];
void Add(int n){
    _rep(i,0,n)
    f[0][i]=f[1][i],f[1][i]=f[2][i];
    f[2][0]=1;
    _rep(i,1,n)
    f[2][i]=((f[1][i]+f[1][i-1])%Mod+f[0][i-1])%Mod;
}
void Mul(int n){
    int _n=build(n<<1);
    _for(i,0,3)NTT(f[i],_n,true);
    _for(i,0,_n){
temp[0][i]=1LL*f[1][i]*f[1][i]%Mod,temp[1][i]=1LL*f[0][i]*f[0][i]%Mod;
temp[2][i]=1LL*f[2][i]*f[1][i]%Mod,temp[3][i]=1LL*f[1][i]*f[0][i]%Mod;
        temp[4][i]=1LL*f[2][i]*f[2][i]%Mod;
    }
    _for(i,0,5)NTT(temp[i],_n,false);
    f[0][0]=temp[0][0],f[1][0]=temp[2][0],f[2][0]=temp[4][0];
    _rep(i,1,n){
        f[0][i]=(temp[0][i]+temp[1][i-1])%Mod;
        f[1][i]=(temp[2][i]+temp[3][i-1])%Mod;
        f[2][i]=(temp[4][i]+temp[0][i-1])%Mod;
    }
    _for(i,n+1,_n)f[0][i]=f[1][i]=f[2][i]=0;
}
int main()
{
    init();
    int n=read_int(),k=read_int(),pos=30;
    while(n<(1<<pos))pos--;
    f[1][0]=f[2][0]=f[2][1]=1;
    while(pos--){
        Mul(k);
        if(n&(1<<pos))Add(k);
    }
    _rep(i,1,k)space(f[2][i]);
    return 0;
}
```

分治 FFT

算法简介

$O\left(n\log^2 n\right)$ 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 $f_0=1,f_{i+1}=\sum_{j=0}^{i} f_jg_{i-j}$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 $[lef,mid]$ 对区间 $[mid,rig]$ 的贡献为

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

套用 NTT 可以 $O(n\log n)$ 求出 $\sum_{j=lef}^{mid} f_jg_{i-j},(mid\le i\lt rig)$于是总时间复杂度为 $O\left(n\log^2 n\right)$

```
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;
}
```

算法练习

CF553E

题意

1. 给定一张 n 个点 m 条边的无重边无自环的有向图，你要从 1 号点到 n 号点去
2. 如果你在 t 时刻之后才到达 n 号点，你要交 x 元的罚款
3. 经过每条边需要支付 w_e 费用，且经过该边消耗 k 个单位时间的概率为 $p_{e,k}(1\le k\le t)$

求最优策略下到达点 n 需要花费的最小费用的期望值，数据范围 $n\le 50,m\le 100,t\le 2\times 10^4$。

题解

设 $\text{dp}(i,j)$ 表示走到点 i 且已经花费 j 个单位时间，达到点 n 还需要花费的最小费用的期望值。

$$\text{dp}(u_e,j)=\min \left(w_e+\sum_{k=1}^tp_{e,k}\text{dp}(v_e,j+k)\right)$$

边界条件为 $\text{dp}(u_e,j)=x+\text{dis}(u,n)(j\gt t),\text{dp}(n,j)=0(j\le t)$。

设 $g(e,j)=\sum_{k=1}^tp_{e,k}\text{dp}(v_e,j+k)$。于是有 $\text{dp}(u_e,j)=\min (w_e+g(e,j))$。

考虑分治 FFT。先计算出 $\text{dp}(u_e,\text{mid}+1\sim \text{rig})$。在计算他们对 $g(e,\text{lef}\sim \text{mid})$ 的贡献，再计算出 $\text{dp}(u_e,\text{lef}\sim \text{mid})$。

其中。$j\in [\text{lef},\text{mid}]$。而 $j+k$ 需要不遗漏地覆盖 $[\text{mid}+1,\text{rig}]$。于是 $k\in [1,\text{rig}-\text{lef}]$。

记 $a_i=p_{e,\text{rig}-\text{lef}-i},b_i=\text{dp}(v_e,i+\text{mid}+1)$。于是

$$g(e,j)\gets \sum p_{e,k}\text{dp}(v_e,j+k)=\sum a_{\text{rig}-\text{lef}-k}b_{k+j-\text{mid}-1}$$

递归边界为 $\text{lef}=\text{rig}$。此时用 $g(e,j)$ 更新 $\text{dp}(u_e,j)$。

由于没有必要通过分治计算出 $\text{dp}(e_u,t+1\sim 2t)$。故可以分治前处理。

总时间复杂度 $O(mt\log^2 t)$。

```
const int MAXN=55,MAXM=105,MAXT=2e4+5;
const double pi=acos(-1.0),Inf=1e9;
struct complex{
    double x,y;
    complex(double x=0.0,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[MAXT<<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;
}
complex t1[MAXT<<3],t2[MAXT<<3];
struct Edge{
    int u,v;
    double w;
}edge[MAXM];
int edge_cnt;
double
dp[MAXN][MAXT<<1],dp2[MAXM][MAXT<<1],dis[MAXN][MAXN],p[MAXM][MAXT<<1];
void cal(int lef,int rig){
    int mid=lef+rig>>1;
    _for(k,0,edge_cnt){
        int n1=rig-lef,n2=rig-mid,n=build(n1+n2-2);
        _for(i,0,n1)t1[i].x=p[k][n1-
i],t1[i].y=0.0;_for(i,n1,n)t1[i].x=t1[i].y=0.0;
_for(i,0,n2)t2[i].x=dp[edge[k].v][i+mid+1],t2[i].y=0.0;_for(i,n2,n)t2[i].x=
t2[i].y=0.0;
        FFT(t1,n,1);FFT(t2,n,1);
        _for(i,0,n)t1[i]=t1[i]*t2[i];
        FFT(t1,n,-1);
        _rep(i,lef,mid)dp2[k][i]+=t1[i+rig-lef-mid-1].x;
    }
}
void cdq(int lef,int rig){
    int mid=lef+rig>>1;
    if(lef==rig){
        _for(i,0,edge_cnt)
        dp[edge[i].u][mid]=min(dp[edge[i].u][mid],dp2[i][mid]+edge[i].w);
        return;
    }
    cdq(mid+1,rig);
```

```
    cal(lef,rig);
    cdq(lef,mid);
}
int main()
{
    int n=read_int(),m=edge_cnt=read_int(),t=read_int();
    double x=read_int();
    _rep(i,1,n)_rep(j,1,n)dis[i][j]=Inf;
    _rep(i,1,n)dis[i][i]=0.0;
    _for(i,0,m){
        edge[i].u=read_int(),edge[i].v=read_int(),edge[i].w=read_int();
        _rep(j,1,t)p[i][j]=read_int()/1e5;
        dis[edge[i].u][edge[i].v]=edge[i].w;
    }
    _rep(k,1,n)_rep(i,1,n)_rep(j,1,n)
    dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);
    _rep(i,0,t)dp[n][i]=0;_rep(i,t+1,t<<1)dp[n][i]=x;
    _for(i,1,n){
        _rep(j,0,t)dp[i][j]=Inf;
        _rep(j,t+1,t<<1)dp[i][j]=x+dis[i][n];
    }
    cal(0,t<<1);
    cdq(0,t);
    printf("%.10lf",dp[1][0]);
    return 0;
}
```

多项式求逆

洛谷p4238

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 $f(x)f_0^{-1}(x)\equiv 1\pmod {x^{\lceil \frac n2\rceil}}$

由于 $f(x)f^{-1}(x)\equiv 1\pmod {x^n}$显然有 $f(x)f^{-1}(x)\equiv 1\pmod {x^{\lceil \frac n2\rceil}}$

于是 $f^{-1}(x)-f_0^{-1}(x)\equiv 0\pmod {x^{\lceil \frac n2\rceil}}$

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 $T(n)=T\left(\frac n2\right)+O(n\log n)$，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 polyinv(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 polyinv(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;
}
```

多项式开根

洛谷p5277

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 $f_0^2(x)\equiv g(x)\pmod {x^{\lceil \frac n2\rceil}}$

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 $\left(\cfrac {f_0^2(x)+g(x)}{2f_0^2(x)}\right)^2 \equiv g(x)\pmod {x^n}$

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

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 polysqrt(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

洛谷p4725

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

$$\mathrm{d}(\ln f(x))\equiv \frac {f^{\prime}(x)}{f(x)}\mathrm{d}x\pmod {x^n}$$

$$\ln f(x)-\ln f(0)\equiv \int_0^x f^{\prime}(t)f^{-1}(t)\mathrm{d}t\pmod {x^n}$$

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

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

```
int inv_f[MAXN<<2];
void polyln(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 $g(f_0(x))\equiv 0\pmod {x^{\lceil \frac n2\rceil}}$。

将 $g(x)$ 在 $f_0(x)$ 处泰勒展开，有

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

同时有 $x^{\lceil \frac n2\rceil}\mid (f(x)-f_0(x))$，于是有 $(f(x)-f_0(x))^i\equiv 0\pmod {x^n}(i\ge 2)$。

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

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

准确来说这里把 $f_0(x)$ 当成了变元 y，$g^{\prime}(f_0(x))=\cfrac {\partial g}{\partial y}(y,x)$。

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

多项式指数函数

洛谷p4726

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 $g(F(x))\equiv \ln F(x)-f(x)\equiv 0\pmod {x^n}$。

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

```
int ln_g[MAXN<<2];
void polyexp(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;
}
```

多项式快速幂

洛谷p5273

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_tx^t 然后提取出 a_tx^t 得到下式

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

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 $k\ln \cfrac{f(x)}{a_tx^t}$ 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 polypow(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;
}
```

多项式除法

洛谷p4512

算法简介

给定 $f(x),g(x)$，不妨记 $\text{deg}(f)=n,\text{deg}(g)=m,n\gt m$，$O(n\log n)$ 时间内求 $q(x),r(x)$ 满足 $$f(x)=q(x)g(x)+r(x)\left(\text{deg}(q)=n-m,\text{deg}(r)\lt m\right)$$

算法实现

构造函数 $f^{R}(x)=x^{\text{deg}(f)}f(\frac 1x)$，易知 $f^{R}(x)$ 与 $f(x)$ 系数恰好反转，可以 $O(n)$ 相互转化。

根据已知，有

$$f(\frac 1x)=q(\frac 1x)g(\frac 1x)+r(\frac 1x)$$

将上式两边同时乘以 x^n，有

$$f^{R}(x)=q^{R}(x)g^{R}(x)+x^{n-{\text{deg}(r)}}r^{R}(x)$$

由于 $n-{\text{deg}(r)}\ge n-m+1$，于是有

$$f^{R}(x)\equiv q^{R}(x)g^{R}(x)\pmod {x^{n-m+1}}$$

于是可以利用多项式求逆求出 $q^{R}(x)$，然后据此求出 $q(x),r(x)$。

```
int temp1[MAXN<<2],temp2[MAXN<<2];
void polydiv(int *f,int *g,int *q,int *r,int n,int m){
    _for(i,0,n)temp1[i]=f[n-1-i];
    _for(i,0,m)temp2[i]=g[m-1-i];
    ployinv(temp2,q,n-m+1);
    int N=build(2*n-m-1);
    NTT(q,N,1);NTT(temp1,N,1);
    _for(i,0,N)q[i]=1LL*q[i]*temp1[i]%Mod;
    NTT(q,N,-1);
    _for(i,0,N)temp1[i]=temp2[i]=0;
    for(int i=0,j=n-m;i<j;i++,j--)swap(q[i],q[j]);
    _for(i,n-m+1,N)q[i]=0;
    _for(i,0,n-m+1)temp1[i]=q[i];
    _for(i,0,m)temp2[i]=g[i];
    N=build(n-1);
    NTT(temp1,N,1);NTT(temp2,N,1);
    _for(i,0,N)temp1[i]=1LL*temp1[i]*temp2[i]%Mod;
    NTT(temp1,N,-1);
    _for(i,0,m-1)r[i]=(f[i]+Mod-temp1[i])%Mod;
    _for(i,0,N)temp1[i]=temp2[i]=0;
}
```

多项式板子汇总

```
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(const int *f,int _n,const int *g,int _m,int *q,int *r){
        static int temp[MAXN<<2];
        _for(i,0,_m)temp[i]=g[_m-1-i];
        Inv(temp,q,_n-_m+1);
        _for(i,0,_n)temp[i]=f[_n-1-i];
        Mul(q,_n-_m+1,temp,_n,_n-_m+1);
        for(int i=0,j=_n-_m;i<j;i++,j--)swap(q[i],q[j]);
        _for(i,0,_m)r[i]=g[i];_rep(i,0,_n-_m)temp[i]=q[i];
        Mul(r,_m,temp,_n-_m+1,_m);
        _for(i,0,_m)r[i]=(f[i]+Mod-r[i])%Mod;
    }
    void Ln(const int *f,int *g,int _n){
        static int temp[MAXN<<2];
        Inv(f,g,_n);
        _for(i,1,_n)temp[i-1]=1LL*f[i]*i%Mod;
        temp[_n-1]=0;
        Mul(g,_n,temp,_n-1,_n);
        for(int i=_n-1;i;i--)g[i]=1LL*g[i-1]*quick_pow(i,Mod-2)%Mod;
        g[0]=0;
    }
    void Exp(const int *f,int *g,int _n){
        static int temp[MAXN<<2];
        if(_n==1)return g[0]=1,void();
        Exp(f,g,(_n+1)>>1);
        _for(i,(_n+1)>>1,_n)g[i]=0;
        Ln(g,temp,_n);
        temp[0]=(1+f[0]-temp[0])%Mod;
        _for(i,1,_n)temp[i]=(f[i]-temp[i])%Mod;
        Mul(g,(_n+1)>>1,temp,_n,_n);
    }
    void Pow(const int *f,int *g,int _n,int k1,int k2){
        static int temp[MAXN<<2];
        int pos=0,posv;
        while(!f[pos]&&pos<_n)pos++;
        if(1LL*pos*k2>=_n){
            _for(i,0,_n)g[i]=0;
            return;
```

```
        }
        posv=quick_pow(f[pos],Mod-2);
        _for(i,pos,_n)g[i-pos]=1LL*f[i]*posv%Mod;
        _for(i,_n-pos,_n)g[i]=0;
        Ln(g,temp,_n);
        _for(i,0,_n)temp[i]=1LL*temp[i]*k1%Mod;
        Exp(temp,g,_n);
        pos=pos*k2,posv=quick_pow(posv,1LL*k2*(Mod-2)%(Mod-1));
        for(int i=_n-1;i>=pos;i--)g[i]=1LL*g[i-pos]*posv%Mod;
        _for(i,0,pos)g[i]=0;
    }
    void Sqrt(const int *f,int *g,int _n){
        static int temp1[MAXN<<2],temp2[MAXN<<2];
        if(_n==1)return g[0]=quick_pow(G,bsgs(3,f[0])/2),void();
        Sqrt(f,g,(_n+1)>>1);
        _for(i,(_n+1)>>1,_n)g[i]=0;
        _for(i,0,_n)temp1[i]=f[i];
        Inv(g,temp2,_n);
        Mul(temp1,_n,temp2,_n);
        int div2=quick_pow(2,Mod-2);
        _for(i,0,_n)g[i]=1LL*(g[i]+temp1[i])*div2%Mod;
    }
}
```

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=1597914058

Last update: **2020/08/20 17:00**