G. Game of Death

题意

一个游戏，有 n 名玩家。每个玩家等概率选择一名除自己以为的人进行射击，射击的命中率为 p。

对每个 $k=1\sim n$ 询问最后存活 k 人的概率。

题解

设 $f(k)$ 表示固定 k 个人的集合，这 k 个人全部死亡的概率。$g(k)$ 表示固定 k 个人的集合，死亡的人是这个集合的子集的概率。

首先考虑计算 $g(k)$。事实上可以把所有人分成两种人，一种是在这个 k 人集合中的人，记为 A 类人。另一种记为 B 类人。

于是只需要保证不杀死 B 类人即可。于是对 A 类人，这个概率为 $1-\frac {p(n-k)}{n-1}$。而对于 B 类人，这个概率为 $1-\frac {p(n-1-k)}{n-1}$。

于是有

$$ g(k)=\left(1-\frac {p(n-k)}{n-1}\right)^k\left(1-\frac {p(n-1-k)}{n-1}\right)^{n-k} $$

同时有 $g(k)=\sum_{i=0}^k {k\choose i}f(i)$。根据二项式反演，得 $$ f(k)=\sum_{i=0}^k (-1)^{k-i}{k\choose i}g(i)=k!\sum_{i=0}^k\frac{(-1)^{k-i}}{(k-i)!}\frac{g(i)}{i!} $$ 卷积计算即可，最终 k 人死亡的答案为 ${n\choose k}f(k)$。时间复杂度 $O(n\log n)$。

```
const int MAXN=3e5+5,mod=998244353;
int quick_pow(int n,int k){
    int ans=1;
    while(k){
        if(k&1)ans=1LL*ans*n%mod;
        n=1LL*n*n%mod;
        k>>=1;
    }
    return ans;
}
namespace Poly{
    const int G=3,Mod=998244353;
    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 n=build(_n+_m-2);
        _for(i,_n,n)f[i]=0;_for(i,_m,n)g[i]=0;
        NTT(f,n,1);NTT(g,n,1);
        _for(i,0,n)f[i]=1LL*f[i]*g[i]%Mod;
        NTT(f,n,0);
    }
}
int frac[MAXN],invf[MAXN];
int C(int n,int m){
    return 1LL*frac[n]*invf[m]%mod*invf[n-m]%mod;
}
int f[MAXN<<2],g[MAXN<<2];
int main()
{
    Poly::init();
    int n=read_int(),a=read_int(),b=read_int();
    int p=1LL*a*quick_pow(b,mod-2)%mod;
    frac[0]=1;
    _for(i,1,MAXN)frac[i]=1LL*frac[i-1]*i%mod;
    invf[MAXN-1]=quick_pow(frac[MAXN-1],mod-2);
    for(int i=MAXN-1;i;i--)
```

```
    invf[i-1]=1LL*invf[i]*i%mod;
    int div=quick_pow(n-1,mod-2);
    _rep(i,0,n){
        LL t1=1-1LL*p*(n-i)%mod*div;
        LL t2=1-1LL*p*(n-1-i)%mod*div;
        g[i]=1LL*quick_pow(t1%mod,i)*quick_pow(t2%mod,n-i)%mod;
        g[i]=1LL*g[i]*invf[i]%mod;
        f[i]=(i&1)?-invf[i]:invf[i];
    }
    Poly::mul(f,n+1,g,n+1);
    _rep(i,0,n)
    f[i]=1LL*f[i]*frac[i]%mod*C(n,i)%mod;
    _rep(i,0,n)
    enter(f[n-i]);
    return 0;
}
```