Atcoder Rugular Contest 124

比赛链接

D - Yet Another Sorting Problem

题意

给定一个 $n+m$ 的排列，每次可以选取一个位置位于 $1\sim n$ 的元素和一个位置位于 $n+1\sim n+m$ 的元素交换位置。

问使得排列有序的最小操作次数。

题解

将排列分解成若干置换环，则对于每个交换操作，等价于选取 $1\le i\le n,n+1\le j\le n+m$，将 $i\to p_i,j\to p_j$ 变为 $i\to p_j,j\to p_i$。

不难发现如果 i,j 属于同一个置换环则环分裂，否则两个环合并。

同时将前 n 个位置染黑，后 m 个位置染白，于是每次操作需要选中一个黑点和一个白点进行操作。

设 A_i 表示第 i 次操作后大小至少为 2 的纯黑色环个数，B_i 表示第 i 次操作后大小至少为 2 的纯白色环个数，C_i 表示第 i 次操作后置换环个数。

定义势能函数 $f(i)=n+m-C_i+2\max(A_i,B_i)$，不难发现 $|C_{i+1}-C_i|=1,|A_{i+1}-A_i|\le 1,|B_{i+1}-B_i|\le 1$。

同时有 $(C_{i+1}-C_i)(A_{i+1}-A_i)\ge 0,(C_{i+1}-C_i)(B_{i+1}-B_i)\ge 0$，于是有 $f(i+1)\ge f(i)+1$。

终态 $A_z=B_z=0,C_z=n+m$，于是有 $f(Z)=0$，最小操作次数不小于 $f(0)-f(Z)=n+m-C_0+2\max(A_0,B_0)$。

下面证明下界可以取到：

首先如果 $A_i\gt 0,B_i\gt 0$，可以选取一个纯黑环和一个纯白环合并，这样 $A_{i+1}-A_i=B_{i+1}-B_i=C_{i+1}-C_i=-1, f(i+1)=f(i)-1$。

若 $A_i\gt 0,B_i=0$，可以选取一个纯黑环和一个含白点的环合并，这样 $\max(A_{i+1},B_{i+1})-\max(A_i,B_i)=-1,f(i+1)=f(i)-1$。

$A_i=0,B_i\gt 0$ 类似处理。最后考虑 $A_i=0,B_i=0$ 的情况，不妨任取一个大小不为 1 的置换环，假设环上黑点不少于白点。

可以找到 i,p_i 满足 i 是白点且 p_i 是黑点，交换位置 i,p_i 的元素等价于从环上单独拆出 p_i。

这样环上黑点数减一，但显然不会形成大小超过 1 的纯白环。于是 A,B 不变，$C_{i+1}=C_i-1,f(i+1)=f(i)-1$。

```
const int MAXN=1e5+5;
int n,m,a[MAXN<<1],vis[MAXN<<1];
int cnt1,cnt2;
void dfs(int pos){
    if(vis[pos])return;
    vis[pos]=true;
    if(pos<=n)cnt1++;
    else
    cnt2++;
    dfs(a[pos]);
}
int main(){
    n=read_int(),m=read_int();
    _rep(i,1,n+m)
    a[i]=read_int();
    int ans=n+m,s1=0,s2=0;
    _rep(i,1,n+m){
        if(!vis[i]){
            ans--;
            cnt1=cnt2=0;
            dfs(i);
            if(cnt1>=2&&cnt2==0)
            s1++;
            else if(cnt2>=2&&cnt1==0)
            s2++;
        }
    }
    enter(ans+2*max(s1,s2));
    return 0;
}
```

E - Pass to Next

题意

给定 n 个人围成一个环，第 i 个人初始时有 a_i 个球，接下来第 i 个人选中 $0\le k\le a_i$ 个球传给第 $i+1$ 个人。

注意第 n 个人把球给第一个人，且所有人同时传球。设一次操作后第 i 个人手上有 b_i 个球，则得到序列 B

对所有可能序列 B先进行一次去重，然后定义每个序列权值为 $\prod_{i=1}^n b_i$求余下所有序列的权值和。

题解

设每个人的传球数为 c_i，不难发现 $\min(c_i)\gt 0$ 时所有 c_i 减一最终得到的序列 B 不变。

于是不妨假设 $\min(c_i)=0$，不难发现在此基础上不同的序列 C 对应不同的序列 B。

利用容斥，答案就等于 c_i 不限制时产生的所有序列 B 的权值和减去 c_i 强制都大于 0 时产生的所有序列 B 的权值和。

接下来考虑如何计算 c_i 无限制时的答案。首先不考虑环的情况，设 $\text{dp}(i,j)$ 表示第 $c_i=j$ 时只考虑前 i 个人产生的序列的权值和，于是有

$$ \text{dp}(i,j)=\sum_{k=0}^{c_{i-1}}\text{dp}(i-1,k)(c_i+k-j) $$

考虑加速 dp，设 $s_1(i)=\sum_{j=0}^{c_i}\text{dp}(i,j),s_2(i)=\sum_{j=0}^{c_i}j\times \text{dp}(i,j)$，于是有

$$ \text{dp}(i,j)=(c_i-j)s_1(i-1)+s_2(i-1)\\
s_1(i)=\sum_{j=0}^{c_i}\text{dp}(i,j)=\sum_{j=0}^{c_i}((c_i-j)s_1(i-1)+s_2(i-1))=(c_is_1(i-1)+s_2(i-1))\sum_{j=0}^{c_i}1-s_1(i-1)\sum_{j=0}^{c_i}j\\
s_2(i)=\sum_{j=0}^{c_i}j\times \text{dp}(i,j)=\sum_{j=0}^{c_i}j((c_i-j)s_1(i-1)+s_2(i-1))=(c_is_1(i-1)+s_2(i-1))\sum_{j=0}^{c_i}j-s_1(i-1)\sum_{j=0}^{c_i}j^2 $$

于是上式可以 $O(n)$ 递推。最后考虑环的处理，不难发现 b_i 的贡献可以分解为本来属于自己的球和来自别人的球。

于是答案可以分解 $\prod_{i=1}^n b_i=(b_1-c_n)\prod_{i=2}^n b_i+c_n\prod_{i=2}^n b_i$。

对于 $(b_1-c_n)\prod_{i=2}^n b_i$，由于只考虑第一个人自己的球，于是有 $\text{dp}(1,i)=a_1-i$，最后贡献为 $\sum_{i=0}^{a_n}\text{dp}(n,i)=s_1(n)$。

对于 $c_n\prod_{i=2}^n b_i$，由于只考虑第 n 个人传给第一个人的球，于是有 $\text{dp}(1,i)=1$，最后贡献为 $\sum_{i=0}^{a_n}i\times \text{dp}(n,i)=s_2(n)$。

关于 c_i 强制都大于 0 的情况，无非是部分公式的下标从 1 开始，计算过程与上述情况类似。

```
const int MAXN=1e5+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;
}
int inv2,inv6,a[MAXN],dp[2][MAXN];
int pre1(int n){
    return 1LL*n*(n+1)%mod*inv2%mod;
}
int pre2(int n){
```

```
    return 1LL*n*(n+1)%mod*(2*n+1)%mod*inv6%mod;
}
void calc(int n,int s1,int s2,bool zero){
    dp[0][1]=s1,dp[1][1]=s2;
    _rep(i,2,n){
dp[0][i]=((1LL*a[i]*dp[0][i-1]+dp[1][i-1])%mod*(a[i]+zero)-1LL*dp[0][i-1]*p
re1(a[i]))%mod;
dp[1][i]=((1LL*a[i]*dp[0][i-1]+dp[1][i-1])%mod*pre1(a[i])-1LL*dp[0][i-1]*pr
e2(a[i]))%mod;
    }
}
int solve(int n,bool zero){
    calc(n,a[1]+zero,pre1(a[1]),zero);
    int t=dp[1][n];
    calc(n,(1LL*(a[1]+zero)*a[1]-pre1(a[1]))%mod,(1LL*pre1(a[1])*a[1]-
pre2(a[1]))%mod,zero);
    return (t+dp[0][n])%mod;
}
int main(){
    int n=read_int();
    _rep(i,1,n)
    a[i]=read_int();
    inv2=quick_pow(2,mod-2);
    inv6=quick_pow(6,mod-2);
    int ans=(solve(n,true)-solve(n,false))%mod;
    ans=(ans+mod)%mod;
    enter(ans);
    return 0;
}
```

F - Chance Meeting

题意

给定 $n\times m$ 的网格，A 开始位于 $(1,1)$，可以从 (r,c) 到达 $(r+1,c)$ 或 $(r,c+1)$，最终目的地为 (n,m)。

B 开始位于 $(n,1)$，可以从 (r,c) 到达 $(r-1,c)$ 或 $(r,c+1)$，最终目的地为 $(1,m)$。

每个时刻只有一个人能移动，问两个人恰好相遇一次的方案。两个方案不同当且仅当某一时刻移动的人物不同或人物的移动方向不同。

题解

为方便计算，令 $n\gets n-1,m\gets m-1$，坐标从 0 开始计算，B 选择 $(r,c)\to (r-1,c)$ 等效于 A 选择 $(r,c)\to (r+1,c)$。

于是 A 最终目的地变为 $(2n,m)$□B 最终目的地变为 (n,m)□B 只在横线运动，于是可以认为两个人只能在 $(n,i)(0\le i\le m)$ 相遇。

设两人在 (n,i) 相遇，且在到达前不相遇的方案数为 $f(i)$□

由于两人仅相遇一次，所有两人从 (n,i) 出发向终点移动的过程也不相遇，该过程的逆过程可以认为是从起点出发到 $(n,m-i)$ 相遇的过程。

由于将 B 选择 $(r,c)\to (r-1,c)$ 等效于 A 选择 $(r,c)\to (r+1,c)$□所以总方案数还要乘上 ${2n\choose n}$□于是合法方案数为

$$ \sum_{i=0}^m {2n\choose n}f(i)f(m-i) $$

接下来考虑计算 $f(i)$□设 $g(i)$ 表示两个人在 (n,i) 相遇的方案数，于是有 $g(i)={n+2i\choose n}{2i\choose i}$□

$f(i)$ 等于 $g(i)$ 再减去多次相遇的方案。考虑枚举两个人在 (n,i) 相遇前最后一次相遇的位置 j□于是两个人在 $(n,j)\to (n,i)$ 的过程中不相遇。

可以任取一个人先走，假定先走的人的每次移动为 $+1$，后走的人的每次移动为 -1。

于是问题等价于现有 $i-j$ 个 ± 1□要求构造序列，使得序列中间部分前缀和大于 0，且序列开头是 $+1$，序列末尾是 -1。

删去序列头尾，问题等价于 $i-j-1$ 个 ± 1□要求构造序列，使得序列前缀和不小于 0。

上述序列数对应 $C(i-j-1)$□其中 $C(n)=\frac {2n\choose n}{n+1}$ 表示卡特兰数，于是有

$$ f(i)=g(i)-\sum_{j=0}^{i-1}g(j)C(i-j-1) $$

卷积加速即可，时间复杂度 $O(n\log n)$□

```
const int MAXN=2e5+5,MAXV=MAXN*3,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;
}
int frac[MAXV],invf[MAXV];
int C(int n,int m){
    return 1LL*frac[n]*invf[m]%mod*invf[n-m]%mod;
}
int Inv(int n){
    return 1LL*invf[n]*frac[n-1]%mod;
}
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);
    }
}
void Init(){
    int n=MAXV-1;
    frac[0]=1;
    _rep(i,1,n)
```

```
    frac[i]=1LL*frac[i-1]*i%mod;
    invf[n]=quick_pow(frac[n],mod-2);
    for(int i=n;i;i--)
    invf[i-1]=1LL*invf[i]*i%mod;
    Poly::init();
}
int f[MAXN],g[MAXN],A[MAXN<<2],B[MAXN<<2];
int main(){
    Init();
    int n=read_int()-1,m=read_int()-1,ans=0;
    _rep(i,0,m){
        g[i]=1LL*C(n+2*i,n)*C(2*i,i)%mod;
        A[i]=g[i];
        B[i]=1LL*C(2*i,i)*Inv(i+1)%mod;
    }
    Poly::mul(A,m,B,m);
    f[0]=g[0];
    _rep(i,1,m)
    f[i]=(g[i]+1LL*(mod-2)*A[i-1])%mod;
    _rep(i,0,m)
    ans=(ans+1LL*f[i]*f[m-i])%mod;
    ans=1LL*ans*C(2*n,n)%mod;
    enter(ans);
    return 0;
}
```