比赛链接

题解

A. 序列

题意

给定一个 $1\sim n$ 的排列 a，该排列的每个子序列 $s=(s_1,s_2\cdots s_p)$ 对 k 的贡献为 $\sum_{i=1}^{p-1}[s_i\lt k\lt s_{i+1}]+[s_i\gt k\gt s_{i+1}]$。

对 $k=1\sim n$，问所有子序列对 k 的贡献总和。

题解

考虑所有子序列中 $(a_i,a_j)(i\lt j)$ 相邻的情况，易知共有 $2^{n-j+i+1}$ 个子序列满足条件。

于是 $(a_i,a_j)(i\lt j)$ 对 $k\in (\min(a_i,a_j),\max(a_i,a_j))$ 的贡献为 $2^{n-j+i+1}$，考虑枚举 (i,j) 然后差分计算贡献，于是得到 $O(n^2)$ 暴力做法。

接下来考虑优化，不妨假设每对 $(a_i,a_j)(i\lt j)$ 对所有 k 产生 $2^{n-j+i+1}$ 的正贡献，然后对 $k\in [1,\min(a_i,a_j)]\bigcup [\max(a_i,a_j),n]$ 产生负贡献。

正贡献总和不难发现为 $\sum_{i=1}^{n-1} (n-i)2^{n-i-1}$，接下来考虑计算负贡献总和。

考虑从 $1\sim n$ 枚举 j，树状数组维护 $1\le i\lt j$ 的贡献和，即 $c_{a_i}=2^ia_i(i\lt j)$。

于是 j 对 $[1,a_j]$ 产生负贡献为 $2^{n-j}\sum_{i=1}^{a_j} c_i$，对 $[a_j,n]$ 产生负贡献为 $2^{n-j}\sum_{i=a_j}^{n} c_i$。

同理，再从 $n\sim 1$ 枚举 i 计算负贡献。最后用正贡献总和减去负贡献总和即可得到答案，时间复杂度 $O(n\log n)$。

```
const int MAXN=1e5+5,mod=1e9+7;
struct BIT{
    #define lowbit(x) ((x)&(-x))
    int c[MAXN];
    void add(int pos,int v){
        while(pos<MAXN){
            c[pos]=(c[pos]+v)%mod;
            pos+=lowbit(pos);
        }
    }
    int query(int pos){
        int ans=0;
        while(pos){
            ans=(ans+c[pos])%mod;
```

```
            pos-=lowbit(pos);
        }
        return ans;
    }
    void clear(){mem(c,0);}
}tree1,tree2;
int query(int L,int R){
    int ans=tree1.query(R);
    if(L)ans=(ans-tree1.query(L-1)+mod)%mod;
    return ans;
}
void add(int L,int R,int v){
    tree2.add(L,v);
    tree2.add(R+1,mod-v);
}
int a[MAXN],pw[MAXN],s[MAXN];
int main()
{
    int n=read_int();
    _rep(i,1,n)
    a[i]=read_int();
    pw[0]=1;
    _for(i,1,MAXN)
    pw[i]=(pw[i-1]<<1)%mod;
    _rep(i,1,n){
        int v1=query(1,a[i]),v2=query(a[i],n);
        v1=1LL*v1*pw[n-i]%mod;
        v2=1LL*v2*pw[n-i]%mod;
        add(a[i],n,v1);
        add(1,a[i],v2);
        tree1.add(a[i],pw[i-1]);
    }
    _rep(i,1,n)
    s[i]=tree2.query(i);
    tree1.clear();
    tree2.clear();
    for(int i=n;i;i--){
        int v1=query(1,a[i]),v2=query(a[i],n);
        v1=1LL*v1*pw[i-1]%mod;
        v2=1LL*v2*pw[i-1]%mod;
        add(a[i],n,v1);
        add(1,a[i],v2);
        tree1.add(a[i],pw[n-i]);
    }
    _rep(i,1,n)
    s[i]=(s[i]+tree2.query(i))%mod;
    int ss=0;
    _rep(i,1,n-1)
    ss=(ss+1LL*(n-i)*pw[n-i-1])%mod;
    _rep(i,1,n)
```

```
    enter((ss-s[i]+mod)%mod);
    return 0;
}
```

E. 上升下降子序列

题意

问有多少个 $1\sim n$ 的排列可以拆分成一个上升子序列和一个下降子序列。

题解

如果一个排列存在多种拆分，则仅考虑上升序列最长的拆分。

如果存在多个最长的上升子序列，则考虑所有最长上升子序列 $(a_1,a_2\cdots a_m)$ 中 a_m 最大的子序列，如果还有相同的则依次比较 $a_{m-1},a_{m-2}\cdots$

易知，这样任意一个满足条件的排列与拆分一一对应。

设 $\text{dp}(i,j,k,t)$ 表示 $1\sim i$ 的排列满足上升子序列最后一个元素下标为 j 且下降子序列的第一个元素下标为 k，且 $t=0/1$ 表示能否将上升序列最后一个元素加入下降序列的方案数。

如果 $k=i+1$ 则表示下降序列为空。枚举新加入 $i+1$ 的位置 p，设原序列为 $s[1\sim i]$，则新序列为 $s[1\sim p-1],i+1,s[p\sim i]$。

同时加入 $i+1$ 时需要维护上升序列最长且逆向字典序最大的性质。

1. 如果 $p\ge j$，显然应该将 $i+1$ 加入上升序列末尾。
2. 如果 $p=j$，则显然原先上升序列的最后一个元素 s_j 被 $i+1$ 替换，于是需要考虑能否将 s_j 加入下降序列。
3. 如果 $p\lt j$，则只能将 $i+1$ 加入下降序列开头，此时需要满足 $p\le k$。

上述转移的正确性由 $1\sim i+1$ 的排列对应的上升序列最长且逆向字典序最大的拆分一定来自唯一的 $1\sim i$ 的排列对应的上升序列最长且逆向字典序最大的拆分再加上 $i+1$ 这条性质保证，但本人不会证明。

```
const int MAXN=105;
int dp[MAXN][MAXN][MAXN][2],mod;
void add(int &x,int y){
    x+=y;
    if(x>=mod)x-=mod;
}
int main()
{
    int n=read_int();
    mod=read_int();
    dp[1][1][2][1]=1;
    _for(i,1,n)_rep(j,1,i)_rep(k,1,j+1)_for(t,0,2){
```

```
        _rep(p,1,i+1){
            if(p>j)
            add(dp[i+1][p][k<p?k:(k+1)][k>=p],dp[i][j][k][t]);
            else if(p==j){
                if(t){
                    if(j<k)
                    add(dp[i+1][p][j+1][1],dp[i][j][k][t]);
                    else
                    add(dp[i+1][p][k][0],dp[i][j][k][t]);
                }
            }
            else{
                if(p<=k)
                add(dp[i+1][j+1][p][j<k?1:t],dp[i][j][k][t]);
            }
        }
    }
    int ans=0;
    _rep(i,1,n)_rep(j,1,n+1)_for(k,0,2)
    add(ans,dp[n][i][j][k]);
    enter(ans);
    return 0;
}
```

D. 方阵的行列式

题意

给定一个模 998244353 （质数）意义下的 $n\times n$ 的方阵和 Q 个修改操作。每个修改操作都修改方阵中某个元素为一个新的值。在每个修改操作后，输出方阵的行列式。

数据范围：

$1\le n,Q\le 500$

（修改第 x 行第 y 列的元素为 z，$1\le x,y\le n,0\le z<998244353$）

题解

前置知识

1. **Sherman-Morrison formula**: 假设 $A\in\mathbb{R}^{n\times n}$ 是可逆方阵，$u,v\in\mathbb{R}^n$ 是列向量，则 $A+uv^T$ 可逆当且仅当 $1+v^TA^{-1}u\ne 0$，此时有 $$ (A+uv^T)^{-1}=A^{-1}-\frac{A^{-1}uv^TA^{-1}}{1+v^TA^{-1}u} $$ 成立
2. **定义**：在 n 阶行列式中，划去元 a_{ij} 所在的第 i 行与第 j 列的元，剩下的元不改变原来的顺序所构成的 $n-1$ 阶行列式称为元 a_{ij} 的**余子式**，记作 M_{ij}。
3. **定义**：a_{ij} 的**代数余子式**：$A_{ij}=(-1)^{i+j}M_{ij}$。

4. **定理**：当 $|A|\ne 0$ 时，$A^{-1}=\dfrac{1}{|A|}A^*$其中 A^* 的第 (i,j) 元为 A 的第 (j,i) 元的代数余子式 $A_{ji}$$A^*$ 称为 A 的伴随方阵。
5. **展开定理**：行列式 Δ 的值，等于它的任意一行各元分别乘以各自的代数余子式的乘积之和$$ \Delta=\sum_{j=1}^na_{ij}A_{ij}. $$ 注意：在按行展开的公式 $$ \Delta=a_{i1}A_{i1}+\cdots+a_{ij}A_{ij}+\cdots+a_{in}A_{in} $$ 中，第 i 行的各元的代数余子式 $A_{i1},\cdots,A_{in}$ 的值都由 Δ 中第 i 行以外的元决定，与第 i 行各元 $a_{i1},\cdots,a_{in}$ 无关。因此，在行列式 Δ 中将第 i 行各元分别换成任意值 $x_1,\cdots,x_n$得到的行列式 $\Delta(x_1,\cdots,x_n)$ 按第 i 行展开式中的各代数余子式 $A_{i1},\cdots,A_{in}$ 仍与在 Δ 中相同。

算法步骤

1. 高斯消元法求 A^{-1} 和 $|A|$记 $A^{-1}=C=(c_{ij})$复杂度 $O(n^3)$
2. 执行修改操作，将 (x,y) 元修改为 z改变量记为 $delta=z-a_{xy}$改变后的矩阵记为 $A'=(a'_{ij})$
3. 计算修改后的行列式的值 $|A'|$用定理 5 对第 x 行按行展开来求，而由定理 4$A'_{ij}=A_{ij}=|A|c_{ji}$故 $\displaystyle |A'|=\sum_{j=1}^na'_{xj}|A|c_{jx}$
4. 令 $|A|=|A'|$并用定理 1 计算 $(A')^{-1}$令 $u=(0,\cdots,0,1,0,\cdots,0)^T$第 x 个分量为 1，其它分量为 0 的列向量$v=(0,\cdots,0,delta,0,\cdots,0)^T$ 第 y 个分量为 $delta$其它分量为 0 的列向量），即可在 $O(n^2)$ 内算出 $(A')^{-1}=(A+uv^T)=A^{-1}-\dfrac{A^{-1}uv^TA^{-1}}{1+v^TA^{-1}u}$然后令 $C=A^{-1}=(A')^{-1},A=A'$回到步骤 2，进行下一步修改操作，直至修改操作结束。

故总时间复杂度 $O(n^3+Qn^2)$

```
#include<bits/stdc++.h>
using namespace std;
const int N=505,mod=998244353;
int A[N][N],B[N][2*N],C[N][N],D[N][N],E[N][N],F[N][N];
int qpow(int x,int y){
    int ret=1;
    for(;y;x=1ll*x*x%mod,y>>=1)
        if(y&1) ret=1ll*ret*x%mod;
    return ret;
}
void Get_Inverse_1(int n,int& det){
    det=1;
    memset(B,0,sizeof(B));
    for(int i=1;i<=n;i++) for(int j=1;j<=n;j++) B[i][j]=A[i][j];
    for(int i=1;i<=n;i++) B[i][i+n]=1;

    int times=0;
    for(int i=1;i<n;i++){
        if(B[i][i]==0){
            times++;
            int row=-1;
            for(int j=i+1;j<=n;j++)
                if(B[j][i]!=0){
                    row=j;
                    break;
```

```
                }
            if(row==-1) continue;
            for(int k=i;k<=2*n;k++) swap(B[i][k],B[row][k]);
        }
        for(int j=i+1;j<=n;j++){
            int m=1ll*B[j][i]*qpow(B[i][i],mod-2)%mod;
            for(int k=i;k<=2*n;k++){
                B[j][k]=(1ll*B[j][k]-1ll*m*B[i][k]%mod+mod)%mod;
            }
        }
        det=1ll*det*B[i][i]%mod;
    }
    det=1ll*det*B[n][n]%mod;
    if(times&1) det=(mod-det)%mod;
    for(int i=n;i>=1;i--){
        int temp=B[i][i];
        temp=qpow(temp,mod-2);
        B[i][i]=1;
        for(int k=n+1;k<=2*n;k++) B[i][k]=1ll*B[i][k]*temp%mod;
        int temp2=qpow(B[i][i],mod-2);
        for(int j=i-1;j>=1;j--){
            int m=1ll*B[j][i]*temp2;
            for(int k=n+1;k<=2*n;k++){
                B[j][k]=(1ll*B[j][k]-1ll*m*B[i][k]%mod+mod)%mod;
            }
        }
    }
    for(int i=1;i<=n;i++) for(int j=1;j<=n;j++) C[i][j]=B[i][j+n];
}
int C2[N],C3[N];
void Get_Inverse_2(int n,int x,int y,int delta){
    int deno=(1+1ll*C[y][x]*delta%mod)%mod;
    deno=qpow(deno,mod-2);
    for(int i=1;i<=n;i++) C2[i]=C[i][x],C3[i]=C[y][i];
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++)
C[i][j]=(1ll*C[i][j]-1ll*C2[i]*delta%mod*C3[j]%mod*deno%mod+mod)%mod;
    }
}
int main(){

    int n,q;
    scanf("%d%d",&n,&q);
    for(int i=1;i<=n;i++) for(int j=1;j<=n;j++) scanf("%d",&A[i][j]);
    int det;
    Get_Inverse_1(n,det);
    while(q--){
        int x,y,z;
        scanf("%d%d%d",&x,&y,&z);
        int before=A[x][y];
```

```
            A[x][y]=z;
            int ans=0;
            for(int j=1;j<=n;j++){
                ans=(1ll*ans+1ll*A[x][j]*C[j][x]%mod*det%mod)%mod;
            }
            printf("%d\n",ans);
            det=ans;
            int delta=(z-before+mod)%mod;
            Get_Inverse_2(n,x,y,delta);
        }
        return 0;
}
```

赛后总结

jxm感觉有一半的时间在 debug

wza代码能力需要加强