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;
}
```

赛后总结

jxm感觉有一半的时间在 debug