# T1 -- game

### 解法：

## 30pts

一共只有四种情况，暴力枚举进行讨论。

## 60pts

考虑暴力模拟博弈的整个过程，$f[S_1][S_2]$ 表示小 A 已经删除了 $S_1$ 集合的这些行，小 B 已经删除了 $S_2$ 集合的这些列会形成的答案，具体转移时，如果是小 A 的回合，那么枚举下一次删除的行，并对所有后继状态取 $\max$，如果是小 B 的回合，那么枚举下一次删除的列，并对所有后继状态取 $\min$。

DP 模拟博弈。

## 100pts

答案即为每行最小值中的最大值，即 $\max_{1\le i\le n}\min_{1\le j\le n}a_{i,j}$。

设答案为 $x$，先证明答案大于等于 $x$，这个过程是平凡的，先手可以删除除最小值等于 $x$ 的行以外的其余行。

再证明答案小于等于 $x$，我们只需要证明每轮后，后手都能保证每行都有小于等于 $x$ 的数。不妨设先手删除了第一行，则后手找到剩余 $n-1$ 行中各一个小于等于 $x$ 的数，标记其所在列，然后删除未被标记的列即可。

时间复杂度 $O(n^2)$。

我们把每一行中最小值的位置标记出来。

---

# Code

```
#include <bits/stdc++.h>
using namespace std;
const int N=1e3+7;
int n,f[N];
int main(){
  std::ios::sync_with_stdio(0);
```

```
  std::cin.tie(0);
  cin>>n; int D=0;
  for(int i=1;i<=n;i++){
    int R=1e9+7;
    for(int j=1;j<=n;j++) cin>>f[j],R=min(R,f[j]);
    D=max(D,R);
  }
  cout<<D<<'\n';
  return 0;
}
```

# T2 -- permutation

## 解法:

### 30pts

注意到一定不会选择两个相交的区间，否则不如直接选这两个区间的并集，代价更小且可以减小的逆序对数更多。

考虑对于单个序列怎么计算答案。

例如 $3,1,2,6,5,4,8,7,9$，修改区间一定为$[3,1,2][6,5,4][8,7][9]$ 用 $|$ 把所有修改区间隔开，即 $3,1,2|6,5,4|8,7|9$，注意到一个位子为 $|$ 当且仅当前面所有的元素均小于后面所有的元素。

因此计算前缀最大和后缀最小，即可知道最优的划分方案。

### 60pts

上述过程等价于计算有多少个 $i,j,k$，满足 $[i,j]$ 中的最大值小于 $[j+1,k]$ 的最小值。

枚举 $j$，计算所有 $i\in[1,j]$ 中 $[i,j]$ 的最大值 $a[i]$，所有 $k\in[j+1,n]$ 中 $[j+1,k]$ 的最小值 $b[k]$，问题等价于计算有多少个 $a[j]<b[k]$，因为值域为 $O(n)$，可以利用前缀和快速计算，时间复杂度为 $O(n^2)$。

### 100pt

考虑枚举 $[i,j]$ 中的最大值 $p[x]$。

那么此时 $p[j+1]$ 一定为 $x$ 右侧第一个比 $p[x]$ 大的元素。

记录 $j+1$ 右侧第一个比 $p[x]$ 小的元素位置为 $y_1$。

记录 $x$ 左侧第一个比 $p[x]$ 大的元素位置为 $y_2$。

容易发现 $i\in[y_1+1,x]$，$k\in[j+1,y_2-1]$，答案即为所有 $(x-y_1)\times(y_2-j-1)$之和。而 $y_1,y_2,j+1$ 均可以利用数据结构快速计算（例如线段树上二分，离线后利用 set 快速计算），时间复杂度为 $O(n\log n)$。

---

## Code

```
#include<bits/stdc++.h>
using namespace std;
int t,n,a[2000005],id[2000005];
set<int>s1,s2;
bool cmp(int x,int y){
    return a[x]<a[y];
}
inline void solve(){
    scanf("%d",&n);
    for(int i=1;i<=n;i++) scanf("%d",&a[i]);
    long long res=0;
    for(int i=2;i<=n;i++)res+=1ll*(i-1)*(n-i+1);
    for(int i=0;i<=n+1;i++)s1.insert(i);
    s2.insert(0);
```

```
    s2.insert(n+1);
    for(int i=1;i<=n;i++)id[i]=i;
    sort(id+1,id+n+1,cmp);
    for(int i=1;i<=n;i++){
        int b=id[i];
        int a=*(--s1.lower_bound(b)),c=*s1.upper_bound(b);
        if(c!=n+1){
            int d=*s2.upper_bound(c);
            res-=1ll*(b-a)*(d-c);
        }
        s1.erase(b),s2.insert(b);
    }
    cout<<res<<endl;
}
signed main(){
    solve();
    return 0;
}
/*
1.选的任意两个区间进行排序
不会有交集
2.[1|5,4,3,2|6,7,8|10,9|13,12,11]
0+3+2+1+2=5+3=8
13-5=8


一个间隔会被称作一个断点
当且仅当这个间隔前面的所有数都小于这个
间隔后面的所有数
答案:区间长度-间隔个数
对于一个序列，怎么计算它的答案呢?
*/
```

# T3 -- xor

## 解法:

### 60pts

$f[i][j]$ 表示前 $i$ 个数异或值为 $j$ 的方案，和正解关系不大故不过多赘述。

### +20pts

$[0,a_i]$ 可以拆分成若干个长度为 $2^c$ 的段。

例如 $a_i=18$ 可以拆成 $[0,15],[16,17],[18,18]$。

例如 $a_i=22$ 可以拆成 $[0,15],[16,19],[20,21],[22,22]$。

注意到这样拆分后，一个长度为 $2^c$ 的段，其中的每个数二进制下后 $c$ 位取遍了 $[0,2^c)$ 中的所有数，且前面的二进制位均相同。

例如$[16,19]$ 二进制下依次为 $10000,10001,10010,10011$，取遍 $00,01,10,11$ 且前面的二进制位均为 $100xx$。

按照如上方式，我们将每个数分成了 $O(\log V)$ 段，假设第 $i$ 个数取的长度为 $2^{len_i}$，取 $len_i$ 中的最大值 $len_{max}$，会发现这些段异或起来的值可以取遍后 $len_{max}$ 段任取的所有可能，即 $2^{len_{max}}$ 种可能，且每种可能的方案数相同。

例如 $a_1=18,a_2=22$，分别取 $[0,15]$ 和 $[16,19]$ 两段，会发现异或起来的权值为 $[16,31]$ 中的所有权值，且每种权值恰好 $4$ 个。分别取 $[16,17]$ 和 $[16,19]$ 两段，会发现异或起来的权值为 $[0,3]$ 中的所有权值，且每种方案数相同。

因此枚举 $n$ 个数每个数取哪一段，会发现答案拆分成了$O(\log V^n)$ 段，每段的前若干个数固定，后若干个数任取所有情况，因此答案可以看作 $O(\log V^n)$ 个段，提前预处理每个段的答案然后作前缀和，时间复杂度为 $O(\log V^n+q)$。

### 100pts

注意到我们只关心 $len_{max}$ 和前缀的形态，因此可以枚举 $len_{max}$，这时候每个数只有两种可能，取 $len_i=len_{max}$ 或 $len_i<len_{max}$，而由于确定了 $len_{max}$ 以后，每个数后 $len_{max}$ 位我们就不关心了（因为取遍了所有情况），注意到一个性质，所有 $len_i<len_{max}$ 的段忽略后 $len_{max}$ 位后的权值都是一样的。

例如 $len_{max}=4$，对于 $[16,19],[20,21],[22,22]$ 它们忽略掉后 $4$ 位以后均为 $16$。

再例如 $len_{max}=2$，$[20,21],[22,22]$ 忽略掉后 $2$ 位以后均为 $20$。

同时 $len_i<len_{max}$ 的前缀和 $len_i=len_{max}$ 的前缀只有最后一位不一样（前者是 $1$，后者是 $0$）。

因此我们可以断言，确定完 $len_{max}$ 以后，可能的前缀只有至多 $2$ 种，且只在最后一位上可能不一样。

可以通过 DP 处理出前缀的具体形态和方案数，计算答案时枚举这至多 $120$ 段进行答案统计即可。

时间复杂度为 $O((n+q)\log V)$。

## Code

```
#include <bits/stdc++.h>
using namespace std;
const int N=1e6+7,M=70,p=998244353;
int n,m; long long f[N],g[N],F[N][4],G[M][2],L[M][2],R[M][2];
void work(){
    cin>>n>>m;
    memset(G,0,sizeof(G));
    for(int i=1;i<=n;i++) cin>>f[i],f[i]++;
    for(int z=0;z<=61;z++){
        long long Lx=(1ll<<(z+1))-1,ans=0;
        for(int j=1;j<=n;j++){
            g[j]=(f[j]^(f[j]&Lx)),ans^=g[j];
            for(int z=0;z<=2;z++) F[j][z]=0;
        }
        F[0][0]=1;
        for(int j=1;j<=n;j++){
            long long Q=(f[j]&Lx),W=0;
            if(f[j]&(1ll<<z)) Q-=(1ll<<z),W=(1ll<<z)%p; Q%=p,W%=p;
            if(W>0){
                F[j][0]=(1ll*F[j-1][1]*Q)%p;
                F[j][1]=(1ll*F[j-1][0]*Q)%p;
                F[j][2]=(1ll*F[j-1][2]*W+1ll*F[j-1][3]*Q)%p;
                F[j][3]=(1ll*F[j-1][3]*W+1ll*F[j-1][2]*Q)%p;
                if(W>0) F[j][2]=(F[j][2]+F[j-1][0])%p;
                if(W>0) F[j][3]=(F[j][3]+F[j-1][1])%p;
            }
            else{
                F[j][0]=(1ll*F[j-1][0]*Q)%p;
                F[j][1]=(1ll*F[j-1][1]*Q)%p;
                F[j][2]=(1ll*F[j-1][2]*Q)%p;
                F[j][3]=(1ll*F[j-1][3]*Q)%p;
            }
        }
        G[z][0]=F[n][2],L[z][0]=ans,R[z][0]=ans+(1ll<<z)-1;
        G[z][1]=F[n][3],L[z][1]=(ans^(1ll<<z)),R[z][1]=L[z][1]+(1ll<<z)-1;
    }
    while(m--){
        long long l,r,sum=0;
        cin>>l>>r;
        for(int z=0;z<=61;z++){
            for(int ii=0;ii<=1;ii++){
                long long A=max(l,L[z][ii]),B=min(r,R[z][ii]);
                if(A>B) continue;
                sum=(sum+(B-A+1)%p*G[z][ii])%p;
            }
        }
        cout<<sum<<'\n';
    }

}
int main() {
  std::ios::sync_with_stdio(false),cin.tie(0),work();
}
```

# T4 --tree

## 解法:

### 10pts

$2^n$ 枚举所有可能的点集即可。

### 20pts

从边的角度来考虑这两棵树，考虑一个 $2n-2$ 个点的有向图，其中 $n-1$ 个点每个点对应一条 $T_1$ 中的边，$n-1$ 个点每个点对应一条 $T_2$ 中的边。
如果选了 $T_1$ 上的一条边 $(u,v)$，那么就必须选 $T_2$ 上路径 $(u,v)$ 上的所有边。把 $T_1$，$T_2$ 反过来也是一样。这样就建出了一张有向图，一条边 $(x,y)$，表示选了 $x$ 必须选 $y$，对这个图作缩点，我们得知选了一个点以后，必须要选后续的所有点，通过缩点+DAG-DP 可以快速得到选中后的所有答案。

一次询问相当于询问这 $c_i$ 个点在 $T_1,T_2$ 两棵树中，两两组成路径所包含的所有边在图中对应点的权值最小值。

### 100pts

上述过程中连边复杂度过高，可以利用倍增连边优化，将边数约束在 $O(n\log n)$ 级别，做 tarjan 缩点后跑 DP，从而先预处理出每条边的答案，最后询问时相当于问每个点到整个点集 LCA 处路径上的最大值，可以利用倍增求解。

---

## Code

```
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <cstring>

using namespace std;

const int maxn = 100010;
const int maxp = 2*maxn+maxn*40*2;
const int inf = 0x3f3f3f3f;

const int maxk = 20;
int n, m, tot = 0;
int w[maxn], fa_T1[maxn], fa_T2[maxn], rid[maxp];

struct Edge {
    int v, x;
};

struct Graph {
    int l[maxp], mx[maxp], dfn[maxp], low[maxp], stack[maxp], vis[maxp], col[maxp], top, tim, e;
    Edge E[maxn*2*20+maxp];
    Graph() {e = 0; memset(l, -1, sizeof(l));}
```

```
    inline void addEdge(int u, int v) {if (!v) return; E[e].v = v; E[e].x = l[u]; l[u] = e++;}
    void tarjan(int u) {
        dfn[u] = low[u] = ++ tim; vis[u] = 1; stack[++ top] = u;
        for (int p = l[u]; p >= 0; p = E[p].x) {
            int v = E[p].v;
            if (!dfn[v]) {
                tarjan(v);
                low[u] = min(low[u], low[v]);
            } else if (vis[v]) {
                low[u] = min(low[u], dfn[v]);
            }
        }
        if (dfn[u] == low[u]) {
            int t = 0;
            do {
                t = stack[top --];
                col[t] = u;
                vis[t] = 0;
                mx[u] = max(mx[u], mx[t]);
                for (int p = l[t]; p >= 0; p = E[p].x) {
                    int v = E[p].v;
                    if (col[v] && col[v] != u) {
                        mx[u] = max(mx[u], mx[col[v]]);
                    }
                }
            } while (t != u);
        }
    }
    void build() {
        for (int i = 1; i <= tot; i++) {
            if (rid[i]) {
                int j = rid[i];
                if (j && j <= n) {
                    mx[i] = max(w[j], w[fa_T1[j]]);
                } else {
                    j -= n;
                    mx[i] = max(w[j], w[fa_T2[j]]);
                }
            } else mx[i] = -inf;
        }
        for (int i = 1; i <= tot; i++)
            if (!dfn[i])
                tarjan(i);
        for (int i = 1; i <= tot; i++) mx[i] = mx[col[i]];
    }
} G;

int mx[maxn][26];

struct Tree {
    int l[maxn], in[maxn][maxk], fa[maxn][maxk], ind[maxn], dep[maxn], e;
    Edge E[maxn<<1];
    Tree() {e = 0; memset(l, -1, sizeof(l));}
    inline void addEdge(int u, int v) {
```

```
        E[e].v = v; E[e].x = l[u]; l[u] = e++;
        E[e].v = u; E[e].x = l[v]; l[v] = e++;
    }
    void dfs(int u, int f) {
        fa[u][0] = f; in[u][0] = ++ tot; if (u > 1) G.addEdge(in[u][0], ind[u]);
        for (int i = 1; i < maxk; i++) {
            fa[u][i] = fa[fa[u][i-1]][i-1];
            in[u][i] = ++ tot;
            G.addEdge(in[u][i], in[u][i-1]); G.addEdge(in[u][i], in[fa[u][i-1]][i-1]);
        }
        for (int p = l[u]; p >= 0; p = E[p].x) {
            int v = E[p].v;
            if (v != f) {
                dep[v] = dep[u] + 1;
                dfs(v, u);
            }
        }
    }
    int calMax(int u, int v) {
        int ret = -inf;
        if (dep[u] < dep[v]) swap(u, v);
        if (dep[u] > dep[v]) {
            int c = dep[u] - dep[v];
            for (int i = 0; i < maxk; i++) {
                if (c & (1 << i)) {
                    ret = max(ret, mx[u][i]);
                    u = fa[u][i];
                }
            }
        }
        if (u == v) return ret;
        for (int i = 19; i >= 0; i--) {
            if (fa[u][i] != fa[v][i]) {
                ret = max(ret, mx[u][i]);
                ret = max(ret, mx[v][i]);
                u = fa[u][i];
                v = fa[v][i];
            }
        }
        ret = max(ret, mx[u][0]);
        ret = max(ret, mx[v][0]);
        return ret;
    }
    void build_mx(int u, int f) {
        mx[u][0] = G.mx[ind[u]];
        for (int i = 1; i < maxk; i++) mx[u][i] = max(mx[u][i-1], mx[fa[u][i-1]][i-1]);
        for (int p = l[u]; p >= 0; p = E[p].x) {
            int v = E[p].v;
            if (v != f)
                build_mx(v, u);
        }
    }
```

```cpp
} T1, T2;

void link(int s, int u, int v, const Tree &T) {
    if (T.dep[u] < T.dep[v]) swap(u, v);
    if (T.dep[u] > T.dep[v]) {
        int c = T.dep[u] - T.dep[v];
        for (int i = 0; i < maxk; i++) {
            if (c & (1<<i)) {
                G.addEdge(s, T.in[u][i]);
                u = T.fa[u][i];
            }
        }
    }
    if (u == v) return;
    for (int i = 19; i >= 0; i--) {
        if (T.fa[u][i] != T.fa[v][i]) {
            G.addEdge(s, T.in[u][i]);
            G.addEdge(s, T.in[v][i]);
            u = T.fa[u][i]; v = T.fa[v][i];
        }
    }
    G.addEdge(s, T.in[u][0]);
    G.addEdge(s, T.in[v][0]);
}

int main() {
    freopen("tree.in", "r", stdin);
    freopen("tree.out", "w", stdout);
    w[0] = -inf;
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d", &w[i]);
    for (int i = 1; i < n; i++) {
        int u, v; scanf("%d%d", &u, &v);
        T1.addEdge(u, v);
    }
    for (int i = 1; i < n; i++) {
        int u, v; scanf("%d%d", &u, &v);
        T2.addEdge(u, v);
    }
    for (int i = 1; i <= n; i++) {
        T1.ind[i] = ++ tot; rid[tot] = i;
        T2.ind[i] = ++ tot; rid[tot] = i+n;
    }
    T1.dfs(1, 0); T2.dfs(1, 0);
    for (int i = 1; i <= n; i++) {
        fa_T1[i] = T1.fa[i][0];
        fa_T2[i] = T2.fa[i][0];
    }
    for (int i = 2; i <= n; i++) {
        link(T1.ind[i], i, T1.fa[i][0], T2);
        link(T2.ind[i], i, T2.fa[i][0], T1);
    }
    G.build();
    T1.build_mx(1, 0);
    for (int i = 1; i <= m; i++) {
```

```
        int c = 0; scanf("%d", &c);
        int u = 0; scanf("%d", &u);
        int ans = w[u];
        for (int j = 1; j < c; j++) {
            int v = 0; scanf("%d", &v);
            ans = max(ans, T1.calMax(u, v));
        }
        printf("%d\n", ans);
    }
    return 0;
}
```