2023 年牛客多校第五场题解

出题人：YNOI 出题组

1 A Jujubesister

题意：给定长度为 n 的数列 $\{a\}_{i=1}^{n}$，q 次询问区间 $[l,r]$ 上满足 $a_i=a_k>a_j$ 且满足 $l\le i<j<k\le r$ 的三元组 (i,j,k) 数目。$1\le n,q\le 5\times 10^5$，$1\le a_i\le n$。

朴素想法：$[l,r]\to[l,r+1]$，$k=r$ 增加，找在区间范围内的 $a_i=a_{r+1}$，离线去查 $r+1$ 前小于 a_i 的 j 个数，

解法：首先考虑如何快速维护这样的三元组。由于有 $i<j<k$ 的约束不好做，考虑维护每个下标 i 前面有多少下标 j 满足 $1\le j<i$ 且 $a_j<a_i$，用 c_i 数组维护。对于一组满足 $a_i=a_k$ 的 (i,k)，则合法的 j 有 c_j-c_i 个。

(i,j,k) 满足 $i<k$ 且 $j<k$ 满足 $a_i=a_k$，$a_j<a_k$ 的 j 个数。要求 $i<j$ 的个数，等价于（全集-不合法的） c_k-c_i，

考虑莫队维护答案，可以再进行一次前缀和，将以 k 为三元组右端点的 (i,j) 个数再求和（即对 c_k 和同色点的 c_i 做前缀和—— $\sum_{i=1,a_i=a_k}^{k} c_k-c_i$，动态维护区间前缀中同色点数目 cnt_{a_i} 和同色点的和 sum_{a_i}）。这样移动区间的时候，可以利用差分快速求出当前的更新量，可以更快维护三元组数目。

因而整体复杂度为莫队的复杂度 $\mathcal{O}\left(n\sqrt{n}+n\log n\right)$。

```cpp
#include <bits/stdc++.h>
#define fp(i, a, b) for (int i = a, i##_ = b; i <= i##_; ++i)
#define fd(i, a, b) for (int i = a, i##_ = b; i >= i##_; --i)

using namespace std;
using ll = long long;
const int N = 5e5 + 5;
int n, m, B, S, T, a[N], b[N], c[N], cnt[N];
ll ANS, sum[N], ans[N];
struct Q { int l, r, id; } q[N];
void add(int i) { for (; i <= n; i += i & -i) ++c[i]; }
int qry(int i, int w = 0) { for (; i; i -= i & -i) w += c[i]; return w; }
void addL(int x) {
    ANS += sum[a[x]] - (ll)cnt[a[x]] * b[x];
    ++cnt[a[x]], sum[a[x]] += b[x];
}
void addR(int x) {
    ANS += (ll)cnt[a[x]] * b[x] - sum[a[x]];
    ++cnt[a[x]], sum[a[x]] += b[x];
}
void delL(int x) {
    --cnt[a[x]], sum[a[x]] -= b[x];
    ANS -= sum[a[x]] - (ll)cnt[a[x]] * b[x];
}
void delR(int x) {
    --cnt[a[x]], sum[a[x]] -= b[x];
```

```
    ANS -= (ll)cnt[a[x]] * b[x] - sum[a[x]];
}
void Solve() {
    scanf("%d%d", &n, &m);
    B = sqrt(n) + 1, S = n / sqrt(m) + 1, T = n / B + 1;
    fp(i, 1, n) scanf("%d", a + i), b[i] = qry(a[i] - 1), add(a[i]);
    fp(i, 0, m - 1) scanf("%d%d", &q[i].l, &q[i].r), q[i].id = i;
    sort(q, q + m, [&](Q a, Q b) { return a.l / S == b.l / S ? (a.l / S & 1 ? a.r > b.r
: a.r < b.r) : a.l < b.l; });
    int L = 1, R = 0;
    fp(i, 0, m - 1) {
        int ql = q[i].l, qr = q[i].r;
        while (L > ql) addL(--L);
        while (R < qr) addR(++R);
        while (L < ql) delL(L++);
        while (R > qr) delR(R--);
        ans[q[i].id] = ANS;
    }
    fp(i, 0, m - 1) printf("%lld\n", ans[i]);
}
int main() {
    int t = 1;
    // scanf("%d", &t);
    while (t--) Solve();
    return 0;
}
```

2 B Circle of Mistery

题意：考虑长度为 n 的每个排列 $P=\{p_1,p_2,\cdots,p_n\}$，并给定权值数组 $\{w\}_{i=1}^n$ 和权值阈值 k，若 P 中存在一个任意长度的置换环 $\{a_1,a_2,\cdots,a_l\}$ 满足 $\sum\limits_{i=1}^{l} w_{a_i}\geq k$，则考虑统计该排列 P 的逆序对数。问符合条件的排列中最小逆序对数目是多少。$1\leq n\leq 10^3$，$-10^6\leq w_i,k\leq 10^6$。

解法：既然需要逆序对数目最少，那么除了给定置换环，那么其他位置 p_i 一定都是 $p_i=i$ 以减少其他部分产生的逆序对数目。

考虑现在置换环构成的数字，假设由 $a_1,a_2,\cdots,a_l$（$a_1<a_2<\cdots<a_l$）构成，则为了逆序对最少，可以考虑按下面构造：$p_{a_1}=a_2,p_{a_2}=a_3,\cdots,p_{a_{l-1}}=a_l,p_{a_l}=a_1$。此时逆序对数目由 p_{a_l} 与前面 p_{a_i} 构成 $l-1$ 个，然后还有满足 $p_i=i$ 且 $i\in[a_1+1,a_l-1]$ 的数字构成。因而总逆序对数目等于 $2(a_l-a_1)-l+1$。因而可以得到一个转化题意：从 $\{1,2,3,\cdots,n\}$ 中选出 l 个数字 $\{a_1,a_2,\cdots,a_l\}$，满足 $\sum\limits_{i=1}^{l} w_{a_i}\geq k$，求 $2(a_l-a_1)-l+1$ 的最小值。

注意到 $n\leq 10^3$，一个朴素的想法是枚举 a_1，顺推枚举 a_l，

纯暴力：$[a_1,a_l]$ 区间根据 w_i 排序，尽可能选大的直到和大于等于 k。

使用优先队列维护 $[a_1,a_l]$ 中哪些数字选出来可以使得其权值和大于等于 k，如果和确定大于等于 k 则按 $2(a_l-a_1)-l+1$ 更新答案。一个贪心的想法是区间中全部的正数必选，尽量选较大的负数，恰好使得其和大于等于 k。但是如果朴素实现，就会发现在顺推过程中维护一个优先队列维护负数会出现上限 $k-w_{a_l}-w_{a_1}$ 不单调，因而导致优先队列无法快速维护满足条件的负数集合。

这时需要考虑一个性质：边界点 a_1 和 a_l 的 w 值必然严格大于 0。考虑固定 a_1，对于 a_l 和 a_l+1，假定 $w_{a_l}>0$ 而 $w_{a_l+1}\le 0$。如果选定置换环末端点为 a_l+1 而非 a_l，则显然选择 a_l+1 可选数字个数（置换环大小）不会优于 a_l 因为确定多选择一个非正权值 w_{a_l+1}，即 l 不会变得更大，而 $a_l+1>a_l$，这样 $2(a_l-a_1)-l+1$ 一定不会更小。同理对于 a_1 可以证明类似结论。因而区间两端 a_1 和 a_l 的 w 值必然严格大于 0。

这时再考虑原始做法。由于确定了 a_1 和 a_l 为正数，则当固定 a_1 而依次增大 a_l 时，随着遍历区间变大，区间中正权值数目和正权值总和会越来越大，这时可以在区间中选择的非正数也会越来越多，这时就可以用一个对顶堆结构维护所有的非正数权值。复杂度 $\mathcal{O}(n\log n)$。

关于对顶堆更加详细一些的说明：考虑使用一个 `used` 小根优先队列表示当前**列入区间选择**的非正数权值，`deleted` 大根优先队列维护当前区间中**未被选择**的非正数权值。当加入一个新的非正数权值 x 时，先考虑加入 `used` 堆，如果 `deleted` 堆顶比 x 大，则将 x 替换入 `deleted`，`deleted` 堆顶送入 `used`。然后再维护 `used` 堆使得它与正数权值和加起来大于等于 k，可以考虑从 `deleted` 堆顶送入若干个最大的元素，或者 `used` 堆踢出若干最小的。

```cpp
#include <bits/stdc++.h>
#define fp(i, a, b) for (int i = a, i##_ = b; i <= i##_; ++i)
#define fd(i, a, b) for (int i = a, i##_ = b; i >= i##_; --i)

using namespace std;
using ll = long long;
const int N = 1e3 + 5;
int n, k, a[N];
void Solve() {
    scanf("%d%d", &n, &k);
    int sum = 0, mx = -1e9, ans = 1e9;
    fp(i, 1, n) {
        scanf("%d", a + i);
        mx = max(mx, a[i]);
        sum += a[i] > 0 ? a[i] : 0;
    }
    if (mx >= k) return puts("0"), void();
    if (sum < k || (sum == 0 && mx < k)) return puts("-1"), void();
    fp(l, 1, n) {
        if (a[l] <= 0) continue;
        priority_queue<int> q1, q2;
        // q1 : selected non-positive, top = lowest
        // q2 : unselected non-positive, top = biggest
        int s = a[l], cnt = 1;
        for (int r = l + 1; r <= n; ++r) {
            if (a[r] > 0) s += a[r], ++cnt;
            else {
                q2.push(a[r]);
```

```
                while (!q1.empty() && !q2.empty() && -q1.top() < q2.top()) {
                    int x = -q1.top(), y = q2.top();
                    s += x - y, q1.pop(), q2.pop(), q1.push(-y), q2.push(x);
                }
                continue;
            }
            while (!q2.empty() && s + q2.top() >= k)
                q1.push(-q2.top()), s += q2.top(), ++cnt, q2.pop();
            if (s >= k) ans = min(ans, 2 * (r - l) - cnt + 1);
        }
    }
    printf("%d\n", ans);
}
int main() {
    int t = 1;
    // scanf("%d", &t);
    while (t--) Solve();
    return 0;
}
```

3 C Cheeeeen the Cute Cat

题意：给定一个二分图，图左右两部都有 n 个点。$\forall i,j\in[1,n]$，若 $(i,j+n)$ 存在边，则 $(j,i+n)$ 不存在边。使用邻接矩阵给出该图的 $\frac{n(n-1)}{2}$ 条边。问该图的二分图最大匹配数。$1\le n\le 3\times 10^3$。

解法：考虑将连边 $(i,j+n)$ 转化到 (i,j) 连边。考虑原二分图上的匹配转化到这个新图上的一条路径——$(a_1,a_2+n),(a_2,a_3+n),\cdots,(a_{k-1},a_k+n)$ 的 $k-1$ 个点的匹配等价于新图上一条长度为 k 的路径 $a_1\to a_2\to\cdots a_k$。

这时原图转化到一个 n 个点的竞赛图（有向完全图）。由于竞赛图必有哈密顿路径，因而答案至少为 $n-1$。考虑何时答案为 n，即存在哈密顿**回路**。显然一个点数大于等于 3 的强连通分量，必然存在哈密顿回路，也就等价于原图上该子图的完美匹配。因而答案为 n 当且仅当图上每个强连通分量都是大于等于 3。只需要特判掉存在孤立强连通块（大小为 1）的情况即可。

```
#include <bits/stdc++.h>
#define fp(i, a, b) for (int i = a, i##_ = b; i <= i##_; ++i)
#define fd(i, a, b) for (int i = a, i##_ = b; i >= i##_; --i)

using namespace std;
using ll = long long;
const int N = 3e3 + 5;
int n, deg[N];
void Solve() {
    scanf("%d\n", &n);
    fp(i, 1, n) fp(j, 1, n) {
        char c = getchar(); getchar();
        deg[i] += c - '0';
```

```
    }
    sort(deg + 1, deg + n + 1);
    for (int i = 1, j = 0, s = 0; i <= n; ++i) {
        s += deg[i];
        if (s == i * (i - 1) / 2) {
            if (i - j < 3) return printf("%d\n", n - 1), void();
            j = i;
        }
    }
    printf("%d\n", n);
}
int main() {
    int t = 1;
    // scanf("%d", &t);
    while (t--) Solve();
    return 0;
}
```

4 D Cirno's Perfect Equation Class

题意：给定 c,k,n，计算满足 $ka+b=c$ 且满足 $\gcd(a,b)\geq n$ 的正整数对 (a,b) 个数。$1\leq c,k,n\leq 10^9$。

解法：枚举 c 的因子 b，计算出 ka，检查是否可以被 k 整除，然后计算出 a，最后检查 $\gcd(a,b)\geq n$ 即可。复杂度 $\mathcal{O}\left(\sqrt{c}\log c\right)$。

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int q, k, c, n;
    scanf("%d", &q);
    while (q--)
    {
        scanf("%d%d%d", &k, &c, &n);
        vector<int> factor;
        for (int i = 1; 1ll * i * i <= c; i++)
            if (c % i == 0)
            {
                factor.push_back(i);
                if (i * i != c)
                    factor.push_back(c / i);
            }
        int ans = 0;
        for (auto b : factor)
        {
            int left = c - b;
```

```
            if (left % k || left <= 0)
                continue;
            int a = left / k;
            if (__gcd(a, b) >= n)
                ans++;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

5 E Red and Blue and Green

题意：构造一个长度为 n 的排列，使得满足 m 个约束条件：$[l_i, r_i]$ 区间逆序对个数是奇数或偶数。满足这些约束条件的区间包含或不相交。$1 \le n, m \le 10^3$。

解法：首先所有的区间不相交或包含意味着这些区间可以构成一个树形结构。首先考虑使用一个 $[1, n]$ 区间包络所有的大区间，并要求区间 $[l, r]$ 上的数字都在 $[l, r]$ 范围。在这个区间包含树（去掉重复区间）上进行 dfs：

1. 当前节点是叶子节点。如果当前逆序对数目为奇数，则交换区间最开头的两个数字，并返回。如果区间长度为 1，返回 `-1`。
2. 当前节点存在子节点 $\{[l_1, r_1], [l_2, r_2], \cdots, [l_k, r_k]\}$。首先子节点遍历，并统计它们逆序对数目奇偶性，将未被子区间覆盖的区间填上相应数字，即对于当前区间 $[l, r]$，若 $x \in [l, r]$ 不属于任何子区间，$a_x \leftarrow x$。
3. 如果当前所有子区间各自内部逆序对奇偶性与大区间相同，直接返回。
4. 如果当前所有子区间逆序对奇偶性与大区间不同，考虑在第一个区间 $[l_1, r_1]$ 和第一个不在第一个区间的位置进行交换。分为以下几种情况：
   (a) $l_1 = l$。考虑交换 r_1 和 $r_1 + 1$。如果 $r_1 + 1 = l_2$，则找到 $[l_2, r_2]$ 的最小值与 $[l_1, r_1]$ 最大值交换。如果 $r_1 + 1 < l_2$，则直接交换 $r_1 + 1$ 和 $[l_1, r_1]$ 最大值。
   (b) $r_1 = r$。考虑交换 $l_1 - 1$ 和 l_1。找到 $[l_1, r_1]$ 的最大值与 $l_1 - 1$ 交换。

```
#include <bits/stdc++.h>
#define fp(i, a, b) for (int i = a, i##_ = b; i <= i##_; ++i)
#define fd(i, a, b) for (int i = a, i##_ = b; i >= i##_; --i)

using namespace std;
using ll = long long;
const int N = 1e3 + 5;
struct Seg { int l, r, k; };
int n, m, a[N], pos[N], deg[N];
vector<Seg> s;
vector<int> G[N];
map<pair<int, int>, int> mp;
void op(int k) { swap(pos[k], pos[k + 1]); }
void dfs(int u) {
    int w = s[u].k;
```

```
    sort(G[u].begin(), G[u].end(), [&](int a, int b) { return s[a].l < s[b].l; });
    for (auto v : G[u])
        dfs(v), w ^= s[v].k;
    if (!w) return;
    if (G[u].empty()) op(s[u].l);
    else if (s[G[u][0]].l == s[u].l) op(s[G[u][0]].r);
    else op(s[G[u][0]].l - 1);
    // fp(i, 1, n) printf("%d%c", pos[i], " \n"[i == n]);
}
void Solve() {
    scanf("%d%d", &n, &m);
    fp(i, 1, n) pos[i] = i, mp[{i, i}] = 0;
    for (int l, r, k; m--;) {
        scanf("%d%d%d", &l, &r, &k);
        if (mp.count({l, r}) && mp[{l, r}] != k)
            return puts("-1"), void();
        if (mp.count({l, r})) continue;
        mp[{l, r}] = k;
        s.push_back({l, r, k});
    }
    m = s.size();
    sort(s.begin(), s.end(), [](Seg a, Seg b) { return a.r - a.l < b.r - b.l; });
    fp(i, 0, m - 1)
        fp(j, i + 1, m - 1)
            if (s[j].l <= s[i].l && s[i].r <= s[j].r) {
                G[j].push_back(i), ++deg[i];
                break;
            }
    fp(i, 0, m - 1)
        if (!deg[i])
            dfs(i);
    fp(i, 1, n) a[pos[i]] = i;
    fp(i, 1, n) printf("%d%c", a[i], " \n"[i == n]);
}
int main() {
    int t = 1;
    // scanf("%d", &t);
    while (t--) Solve();
    return 0;
}
```

6 G Go to Play Maimai DX

题意：给定长度为 n 的数列 $\{a\}_{i=1}^{n}=\{1,2,3,4\}^n$，问长度最短的区间包含的 $\{1,2,3\}$ 和至少 k 个 4。$1\le k\le n\le 10^5$。

解法：右端点单调右移的时候，左端点必然也单调右移。双指针扫描一下即可。复杂度 $\mathcal{O}(n)$。

```cpp
#include<bits/stdc++.h>
using namespace std;
#define fre(x) freopen(#x".in","r",stdin);freopen(#x".out","w",stdout)
typedef long long ll;
template<typename T>inline void read(T &a){
    char c=getchar();T x=0,f=1;
    while(!isdigit(c)){if(c=='-')f=-1;c=getchar();}
    while(isdigit(c)){x=(x<<1)+(x<<3)+(c^48);c=getchar();}
    a=f*x;
}
inline void write(ll x){
    char P[105];int w=0;
    if(x<0)putchar('-'),x=-x;
    if(x==0)printf("0");
    while(x)P[++w]=x%10+'0',x/=10;
    for(int i=w;i;i--)putchar(P[i]);
}
const int N=1e5+10;
int cnt[N];
int a[N];
int n,k;
bool pd(){
    for(int i=1;i<=3;++i){
        if(!cnt[i])return false;
    }
    if(cnt[4]<k)return false;
    return true;
}
int main()
{
    scanf("%d%d",&n,&k);
    for(int i=1;i<=n;++i)scanf("%d",&a[i]);
    int minlen=n;
    for(int l=1,r=0;r<=n&&l<=n;++l){
        while((!pd())&&r<n){
            cnt[a[++r]]++;
        }
        if(pd())minlen=min(minlen,r-l+1);
        cnt[a[l]]--;
    }
    printf("%d\n",minlen);
```

```
    return 0;
}
```

7 H Nazrin the Greeeeeedy Mouse

题意：给定 n 个奶酪，体积为 a_i，价值为 b_i，只能从 1 至 n 顺序拿，如果要拿第 $i+1$ 个奶酪，第 i 个奶酪必须拿走或者打洞，被打洞的奶酪不能再被拿走。给定 m 个背包，体积为 $\{\text{size}\}_{i=1}^{m}$，每次拿着一个背包从第一个奶酪出发拿奶酪，问 m 个背包下能拿走多少价值的奶酪。$1 \le n \le 200$，$1 \le a_i \le 200$，$1 \le b_i \le 10^5$，$1 \le \text{size}_i \le 200$，$1 \le m \le 10^5$。

解法：不难注意到，每次有效拿背包装奶酪至少会拿一个奶酪走。因而至多只需要 $\min(n, m)$ 个背包即可，并且一定是贪心取最大的背包。并且每个奶酪只有拿走或者不拿走（被打洞），而且还是顺序拿走。

考虑背包：$f_{i,j,k}$ 表示从第 i 个奶酪开始拿，拿到第 j 个奶酪取得体积为 k 的最大价值。一个非常经典的 01 背包：

$$f_{i,j,k} \leftarrow \max(f_{i,j-1,k}, f_{i,j-1,k-a_i} + b_i)$$

维护该背包的复杂度为 $\mathcal{O}(n^2V)$。然后考虑维护前缀 min：$f_{i,j,k} \leftarrow \max(f_{i,j,k-1}, f_{i,j,k})$。

然后再维护一个 dp：$g_{i,j}$ 表示用了 i 个背包，现在已经拿到第 j 个奶酪的最大价值。转移是显然的：

$$g_{i,j} \leftarrow \max_{k \in [0, j-1]} g_{i-1,k} + f_{i,j,\text{size}_i}$$

因而总复杂度 $\mathcal{O}(n^2V + nV)$。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 200;
int a[N + 5], siz[100005], use[N + 5];
int f[N + 5][N + 5][N + 5], b[N + 5], g[N + 5][N + 5];
int main()
{
    int n, m;
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++)
        scanf("%d%lld", &a[i], &b[i]);
    for (int i = 1; i <= n; i++)
    {
        for (int j = i; j <= n; j++)
        {
            for (int w = 0; w <= N; w++)
                f[i][j][w] = f[i][j - 1][w];
            for (int w = a[j]; w <= N; w++)
                f[i][j][w] = max(f[i][j][w], f[i][j - 1][w - a[j]] + b[j]);
        }
        for (int j = 1; j <= n; j++)
            for (int w = 1; w <= N; w++)
                f[i][j][w] = max(f[i][j][w], f[i][j][w - 1]);
```

```
    }
    for (int i = 1; i <= m; i++)
        scanf("%d", &siz[i]);
    int cnt = 0, ans = 0;
    for (int i = max(1, m - n); i <= m; i++)
        use[++cnt] = siz[i];
    for (int i = 1; i <= cnt; i++)
        for (int j = 1; j <= n; j++)
        {
            for (int k = 0; k < j; k++)
                g[i][j] = max(g[i][j], g[i - 1][k] + f[k + 1][j][use[i]]);
            ans = max(ans, g[i][j]);
        }

    printf("%d", ans);
    return 0;
}
```

8 I The Yakumo Family

题意：给定长度为 n 的数列 $\{a\}_{i=1}^{n}$，定义 $f(l,r)=\bigoplus_{i=l}^{r} a_i$，求下式：

$$\sum_{1\le l_1\le r_1\le n}\sum_{r_1<l_2\le r_2\le n}\sum_{r_2<l_3\le r_3\le n} f(l_1,r_1)\times f(l_2,r_2)\times f(l_3,r_3)$$

$1\le n\le 2\times 10^5$，$0\le a_i\le 10^9$。

解法：显然对于本题应该考虑拆位。

首先考虑如何求

$$\sum_{1\le l_1\le r_1\le n} f(l_1,r_1)$$

以拆位视角，仅考虑第 k 个二进制位的贡献。当对 $\{a\}$ 进行前缀异或操作（得到 $\{s\}_{i=0}^{n}$）后，对于所有右端点 r_1 确定的区间 $[x,r_1]$，该区间在该二进制位上贡献为 1 的次数为所有 $s_{x-1}\neq s_{r_1}$ 且满足 $x\le r_1$ 的下标 x 数目，且每次出现对上式的贡献都为 1。因而可以考虑随着 r_1 增大维护数组 P_0/P_1 表示该位上前缀异或和 s_x 是 $0/1$ 的下标 x 次数。

因而综合每一个二进制位，我们可以维护 $\{S_1\}_{i=1}^{n}$ 数组，其中 $S_{1,i}$ 表示区间右端点固定在 i 的区间的异或和的和。这时考虑对 $S_{1,i}$ 进行前缀和操作得到 $\{T_1\}$ 数组，则 $T_{1,n}$ 即为 $\sum_{1\le l_1\le r_1\le n} f(l_1,r_1)$。

接下来考虑求

$$\sum_{1\le l_1\le r_1\le n}\sum_{r_1<l_2\le r_2\le n} f(l_1,r_1)\times f(l_2,r_2)$$

首先还是考虑拆位。考虑当 r_2 固定为 i 时，所有区间 $[x,r_2]$ 如果该位上异或和为 1，必须满足 $s_{x-1}\neq s_{r_2}$ 且满足 $x\le r_2$ 的下标 x，且每个 x 对答案的贡献为 $T_{1,x-1}$——即第一区间的右端点在 $[1,x-1]$ 上。因而可以考虑随着 r_1 增大维护数组 P_0/P_1 表示该位上前缀异或和 s_x 是 $0/1$ 的 $T_{1,x}$ 的和。

这时综合每个二进制位，就可以维护 $\{S_2\}_{i=1}^{n}$ 数组表示第二区间右端点固定在 i 的区间的异或乘积和。

因而对于如果有 k 个不相交区间，也可以类似处理。本题 $k=3$，因而只需要进行三次。总复杂度 $\mathcal{O}(kn\log a_i)$。

```cpp
#include <bits/stdc++.h>
#define fp(i, a, b) for (int i = a, i##_ = b; i <= i##_; ++i)
#define fd(i, a, b) for (int i = a, i##_ = b; i >= i##_; --i)

using namespace std;
using ll = long long;
const int N = 2e5 + 5, M = 30, P = 998244353;
int n, a[N], f[N], s[N];
void Solve() {
    scanf("%d", &n);
    s[0] = 1;
    fp(i, 1, n) scanf("%d", a + i), a[i] ^= a[i - 1], s[i] = 1;
    fp(_, 1, 3) {
        fp(j, 0, M) {
            ll c[2] = {s[0], 0};
            fp(i, 1, n) {
                f[i] = (f[i] + (c[(a[i] >> j & 1) ^ 1] << j)) % P;
                (c[a[i] >> j & 1] += s[i]) %= P;
            }
        }
        s[0] = 0;
        fp(i, 1, n) s[i] = (s[i - 1] + f[i]) % P, f[i] = 0;
    }
    printf("%d\n", s[n]);
}
int main() {
    int t = 1;
    // scanf("%d", &t);
    while (t--) Solve();
    return 0;
}
```