2024 CSP-S 复赛完整版解析（含标程）云斗学院版

T1.决斗

25 pts（$r_i \le 2$）

此时 r_i 只有两种，令 x 表示 1 的个数，y 表示 2 的个数，我们可以让 $s=\min(x,y)$ 个 2 去干掉 s 个 1，即 $\text{ans}=n-s$。

25 pts（数据随机）

由于 n 只有 30，而 $V=10^5$ 很大，且数据随机生成，故极大概率 a_i 互不相同。此时考虑贪心，排序使得 $a_1<a_2<\cdots<a_n$ 时，可依次让每个 a_i 干掉 a_{i-1}，即 $\text{ans}=1$。

100 pts

上面的做法启示我们，要尽可能让每个人充分发挥用处，即干掉一个人再被另一个人干掉。那容易有贪心策略，将其排序后，从小往大枚举 i，维护 k 表示当前未被删除的编号最小的人，若 $a_i>a_k$，则用 a_i 干掉 a_k，令 $k\leftarrow k+1,\ \text{ans}\leftarrow\text{ans}+1$。反之，若$a_i=a_k$，无事发生。

复杂度 $\mathcal{O}(n\log n)$。

std

```cpp
#include <bits/stdc++.h>
using namespace std;
using lint = long long;
const int N = 1e5 + 5;
int n, a[N], p[N], ans = 1;
signed main() {
    // freopen("duel.in", "r", stdin);
    // freopen("duel.out", "w", stdout);
    ios::sync_with_stdio(0), cin.tie(0), cout.tie(0);
    cin>>n;
    for(int i = 1; i <= n; i++) cin>>a[i];
    sort(a + 1, a + n + 1);
    for(int i = 1; i <= n; i++) if(a[ans] < a[i]) ans++;
    cout<<n - (ans - 1)<<endl;
    return 0;
}
```

T2.超速检测

10 pts（$n,m\le 10$）

对不等式变形，将 $\sqrt{v_0^2+2as}\le V$ 两边同时平方得到 $v_0^2+2as\le V^2$，这样避免了浮点数运算导致的精度问题。

对于第一问，枚举每辆车，枚举每个摄像头，判断是否超速即可。

对于第二问，枚举每个摄像头开启或关闭，用第一问的方法判断原来超速的车是否还超速。

令 n, m 同阶，复杂度 $\mathcal{O}(2^n \times n^2)$。

20 pts（$n, m \le 20$）

考虑优化，我们不妨算出每辆车在通过哪个摄像头时速度最大。通过预处理可以在做第二问的时候去掉一个 m。复杂度$\mathcal{O}(2^n \times n)$。

60 pts（$n, m \le 3 \times 10^3$）

容易发现，每辆车的速度是单调的，这也就是说，能检测其超速的摄像头组成了一个区间。我们可以用 $x - v$ 公式二分的处理出每辆车的超速区间。问题转化为，给定 n 条线段，问最少需要多少个点使得每条线段至少覆盖了一个点。

考虑一个贪心，维护每个点被哪些线段覆盖，每次选择被覆盖次数最多的点，并将覆盖它的这些线段的贡献去掉。

复杂度 $\mathcal{O}(n^2)$。

100 pts

考虑贪心，我们希望用一个点能满足尽量多的线段，即将这些线段分成尽可能少的组，使得每组中所有线段的交不为空。由此，不难想到按左端点排序后，依次放入每个线段，维护当前的交，若加入后仍有交，则贪心的加入，显然不劣，否则新开一组。复杂度 $O(n \log n)$。

std

```
#include <bits/stdc++.h>
using namespace std;
using lint = long long;
const int N = 1e5 + 5, INF = 1e9;
lint n, m, L, V, d[N], v[N], a[N], s[N], p[N], l[N], r[N], cnt, ok[N], ans = 1;
bool check(int k, int t) {
    return v[k] * v[k] + a[k] * (p[t] - d[k]) * 2 <= V * V;
}
int work1(int k, int ll, int rr) {
    int ret = 0;
    while(ll <= rr) {
        int mm = (ll + rr) / 2;
        if(check(k, mm)) ll = mm + 1;
        else rr = mm - 1, ret = mm;
    }
    return ret;
}
int work2(int k, int ll, int rr) {
    int ret = 0;
    while(ll <= rr) {
        int mm = (ll + rr) / 2;
        if(check(k, mm)) rr = mm - 1;
```

```
            else ll = mm + 1, ret = mm;
        }
        return ret;
}
vector<pair<int, int>> g;
void solve() {
        cin>>n>>m>>L>>V; cnt = 0, ans = 1, g.clear();
        for(int i = 1; i <= n; i++) cin>>d[i]>>v[i]>>a[i];
        for(int i = 1; i <= m; i++) cin>>p[i];
        for(int i = 1; i <= n; i++)
            if(a[i] >= 0) s[i] = L;
            else s[i] = d[i] - v[i] * v[i] / (a[i] * 2);
                // v * v + 2as = 0  --->  s = -v * v / (2a)
        for(int i = 1; i <= n; i++)
            if(a[i] >= 0) r[i] = m + (d[i] > p[m]);
            else r[i] = lower_bound(p + 1, p + m + 1, d[i]) - p;
        for(int i = 1; i <= n; i++)
            if(r[i] <= m && !check(i, r[i])) cnt++, ok[i] = 1;
            else ok[i] = 0;
        for(int i = 1; i <= n; i++) if(ok[i]) {
            if(a[i] > 0) l[i] = work1(i, lower_bound(p + 1, p + m + 1, d[i]) - p,
r[i]);
            else if(a[i] == 0) l[i] = lower_bound(p + 1, p + m + 1, d[i]) - p;
            else l[i] = r[i], r[i] = work2(i, l[i], upper_bound(p + 1, p + m + 1,
s[i]) - p - 1);
            // cout<<i<<" "<<l[i]<<" "<<r[i]<<endl;
            g.push_back({l[i], r[i]});
        }
        sort(g.begin(), g.end());
        int mn = INF;
        for(auto x : g) {
            int ll = x.first, rr = x.second;
            if(ll <= mn) mn = min(mn, rr);
            else mn = rr, ans++;
        }
        if(g.empty()) ans = 0;
        cout<<cnt<<" "<<m - ans<<endl;
}
signed main() {
        // freopen("detect.in", "r", stdin);
        // freopen("detect.out", "w", stdout);
        ios::sync_with_stdio(0), cin.tie(0), cout.tie(0);
        int t; cin>>t;
        while(t--) solve();
        return 0;
}
```

T3.染色

$\mathcal{O}(T \times 2^n)$ 的暴力可以取得 20 pts。

我们考虑记录 dp 数组 $f_{i,j}$ 表示现在给第 i 个人颜色，上一个异色点的**数值**为 j 的答案的最大值，0 表示没有选过异色点，初始时 $f_{1,0}=0, f_{else}=-\infty$。

如果 $i \neq 1$，分两种情况转移：

1.如果 $j=a_{i-1}$，则可以从 $\max(f_{i-1,a_i}+a_i, \max f_{i-1,k})$ 中转移，这相当于钦定 $i-1$ 为异色点。

2.无论 j 的取值，可以钦定 $i-1$ 为同色点，则上一个异色点与之相同。则有转移 $f_{i,j}=\max f_{i-1,j}+a_i[a_i=a_{i-1}]$。

如果记录的是上一个编号，则是 $\mathcal{O}(n^2)$ 的，如果记录数值，则是 $\mathcal{O}(nv)$ 的。

考虑优化。

发现了这个相当于进行了全局加，单点修改，单点查询，全局求最大值。

显然可以线段树维护。

也可以维护 tag，时间复杂度 $\mathcal{O}(n)$，具体的，维护全局加总和 s，单点修改时令 $a_x=-s$，查询时即为 $s+a_x$。

std

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
const int N = 1e6 + 7;
int dp[N] , u[N] , a[N] , s , M ;
inline int query(int x){
    return s + dp[x] - u[x];
}
inline void solve(){
    memset(dp , -0x3f , sizeof dp);
    memset(u , 0, sizeof u);
    int n;
    cin >> n;
    s = M = 0;
    for(int i = 1; i <= n; ++ i){
        cin >> a[i];
    }
    for(int i = 2; i <= n; ++ i){
        int x = a[i] == a[i - 1] ? a[i] : 0;
        int j = a[i - 1];
        int res = max(query(a[i]) + a[i] , M);
        if(res > query(j) ){
            dp[j] = res;
            u[j] = s + x;
        }
        s += x;
        M += x;
        M = max(M , query(j));
    }
    cout << M << '\n';
```

```
}
signed main(){
    int T;
    cin >> T;
    while(T --) solve();
    return 0;
}
```

T4.擂台游戏

由于每个询问对应的 k 是不同的，所以首先枚举 k，处理所有 $\lceil \log_2 c_i \rceil = k$ 的询问。

首先，我们需要建出此时对局的结构，其实就是一颗满二叉树，笔者使用了线段树的标号方式：叶子节点编号从 2^k 开始到 $2^{k+1}-1$，根节点为 1。

同时，注意到每个询问已知的 a_i 是一段前缀，考虑从前往后依次加入每个 a_i 并实时维护答案。

发现当确定的 a_i 变多时，游戏的可能性变少了，可能获胜的人数也变少。具体地，如果 i 此时已经无法获胜，那么插入更多的 a_i 时，同样不可能获胜，这启发我们在每次插入时，找到所有此时**变为**无法获胜状态的点，减去他们的贡献。

考虑依次确定 a_i 的过程：当确定一个 a_i 时，会使得二叉树上 i 的若干祖先节点（是一段连续的区间）的胜者唯一确定，这一部分可以直接暴力向上判断，如果无法确定，就退出。这样可以做到每次都把一个未确定的点变成唯一确定的，而总点数是 $O(2^k)$ 的，所以均摊是 $O(1)$。

接下来，考虑有哪些选手可能成为最终的胜者。首先。如果这个选手在某一场对局中未能晋级，那么一定不行，这点是显然的，在之前二叉树向上跳父亲的过程中实时维护，需要支持子树删除（一个子树内的点都无法获胜了），可以对每个子树打标记，如果该子树没有被标记过，那么递归到子树内部，否则退出，遍历到叶子节点时，减去他的编号即可。显然，这部分同样是均摊 $O(1)$ 的，代码如下：

```
void remove(int u){
    if(vis[u])return; // already removed
    vis[u]=1;
    if(u<(1<<k))remove(u<<1),remove(u<<1|1);
    else cur-=u-(1<<k)+1; // leaf node
}
```

但是，这样处理后，剩余的选手中仍然有一些选手无法成为最终胜者，就是那些虽然还没有输过，但权值不够大，导致他在未来的某一局中一定无法获胜。这可以 $O(2^k)$ 预处理一遍二叉树上每个节点能够获胜的权值最小值，在插入完 a_i 时，如果权值太小，则直接删掉 a_i。

至此，我们就可以在 $O(2^K)+O(2^{K-1})+\cdots+O(1)=O(2^K)$ 的时间复杂度内解决一组询问，故总的时间复杂度为 $O(T2^K+Tm)=O(T(n+m))$。

需要一些卡常技巧：

- 在二叉树向上跳的过程中，使用非递归时的写法；
- 在处理 2^k 的答案时，可以沿用之前 2^{k-1} 的信息，减少重复的运算。

std

```
#include<bits/stdc++.h>
using namespace std;
using ll=long long;
const int N=1<<17|5,M=N*2;
int T,n,m,k,p,a0[N],a[N],c[N],d[N];
int vis[M],val[M],lim[18][M];
ll cur,ans[N];
void remove(int u){
    if(vis[u])return;
    vis[u]=1;
    if(u<(1<<k))remove(u<<1),remove(u<<1|1);
    else cur-=u-(1<<k)+1;
}
void update(int u,int x){
    if(x<lim[p][u])remove(u);
    for(int id=1;;id++,u>>=1){
        val[u]=x;
        if(u==(1<<k-p))return;
        if(u&1){
            if(d[u>>1]){
                if(x>=id)remove(u^1);
                else x=val[u^1],remove(u);
            }else{
                if(~val[u>>1])return;
            }
        }else{
            if(d[u>>1])return;
            else{
                if(x>=id)remove(u^1);
                else return remove(u);
            }
        }
    }
}
void solve(){
    int l=(1<<p-1)+1,r=1<<p;
    cur+=(r-l+1ll)*(l+r)/2;
    if(!d[1<<k-p]){
        if(val[2<<k-p]>=p){
            val[1<<k-p]=val[2<<k-p];
            remove(2<<k-p|1);
        }else remove(2<<k-p);
    }
    for(int i=1,u=1<<k;i<=(1<<p-1);i++,u++){
        if(a[i]<lim[p][u])remove(u);
    }
    for(int i=l,u=(1<<k)+l-1;i<=r&&i<=n;i++,u++){
        update(u,a[i]),ans[i]=cur;
    }
}
```

```
int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)scanf("%d",&a0[i]);
    for(int i=1;i<=m;i++)scanf("%d",&c[i]);
    for(k=0;(1<<k)<n;k++);
    for(int len=1<<k-1;len;len>>=1){
        static char str[N];
        scanf("%s",str);
        for(int i=0;i<len;i++)d[len|i]=str[i]&1;
    }
    for(p=1;p<=k;p++){
        lim[p][1<<k-p]=0;
        for(int s=1<<k-p,len=1,id=p;id;s<<=1,len<<=1,id--){
            for(int u=s;u<s+len;u++){
                lim[p][u<<1]=max(lim[p][u],d[u]?0:id);
                lim[p][u<<1|1]=max(lim[p][u],d[u]?id:0);
            }
        }
    }
    scanf("%d",&T);
    for(int X[4];T--;){
        for(int i:{0,1,2,3})scanf("%d",&X[i]);
        for(int i=1;i<=n;i++)a[i]=a0[i]^X[i&3];
        fill(vis+1,vis+(1<<k+1),0);
        fill(val+1,val+(1<<k+1),-1);
        ans[1]=1,val[1<<k]=a[1],cur=1;
        for(p=1;p<=k;p++)solve();
        ll res=0;
        for(int i=1;i<=m;i++)res^=i*ans[c[i]];
        printf("%lld\n",res);
    }
    return 0;
}
```