CSP-S 2024 解题报告

[CSP-S 2024] 决斗

基本思路

本题等价于将可重集 $\{r_n\}$ 划分成若干个严格递增的序列。

构造方案：每次从 $\{r_n\}$ 每种数中各取一个构成一个序列。

容易发现，答案即为数量最多的一种数的数量，即众数。

代码实现

```
#include<bits/stdc++.h>
using namespace std;
const int N=100100,M=100000;
int n,a[N],cnt[N],ans=0;
signed main(){
    ios::sync_with_stdio(false),cin.tie(0),cout.tie(0);
    cin>>n;
    for(int i=1;i<=n;i++) cin>>a[i],cnt[a[i]]++;
    for(int i=1;i<=M;i++) ans=max(ans,cnt[i]);
    cout<<ans<<'\n';
    return 0;
}
```

[CSP-S 2024] 超速检测

基本思路

本题对第一问先考虑每辆车超速的位置区间 S_i，对 a_i 分类讨论：

- 当 $a_i = 0$ 时
  - 若 $v_i > V$，$S_i = [d_i, L]$.
  - 若 $v_i \le V$，$S_i = \varnothing$.
- 当 $a_i > 0$ 时
  - 若 $v_i > V$，$S_i = [d_i, L]$.
  - 若 $v_i \le V$，$S_i = (d_i + \frac{V^2 - v_i^2}{a_i}, L]$.
- 当 $a_i < 0$ 时
  - 若 $v_i > V$，$S_i = [d_i, d_i + \frac{V^2 - v_i^2}{a_i})$.
  - 若 $v_i \le V$，$S_i = \varnothing$.

随后二分计算对应的测速仪区间 $[l_i, r_i]$（可以使 $p_0 \leftarrow 0, p_{m+1} \leftarrow L + 1$ 来避免讨论边界情况）。

若 $l_i \ge r_i$，则车 i，会被监测到。

接下来是第二问，考虑计算保留那些测速仪。

考虑贪心，将所有区间 $[l_i, r_i]$ 以 r_i 为第一关键字，以 l_i 为第二关键字排序。

用 i 遍历 $1,2,\cdots,n$，令 $last$ 为当前编号最大的开启的测速仪的编号，cnt 为已开启的测速仪数量：

- 若 $last < l_i$，开启 r_i，令 $last \leftarrow r_i, cnt \leftarrow cnt+1$。
- 否则这辆车已被测到，无需操作。

第二问答案即为 $m-cnt$。

代码实现

```
#include<bits/stdc++.h>
using namespace std;
const int N=100100;
int n,m,L,V,d[N],v[N],a[N],p[N],tot;
struct car{int l,r;}c[N];
int cl(int a,int b){return a/b+1;}
int fl(int a,int b){return (a-1)/b;}
void solve(){
    cin>>n>>m>>L>>V;
    for(int i=1;i<=n;i++) cin>>d[i]>>v[i]>>a[i];
    for(int i=1;i<=m;i++) cin>>p[i];
    p[0]=0,p[m+1]=L+1;tot=0;
    for(int i=1;i<=n;i++){
        int l=0,r=0;
        if(a[i]==0){
            if(v[i]>V) l=d[i],r=L;
            else continue;
        }
        else if(a[i]>0){
            if(v[i]>V) l=d[i],r=L;
            else{
                l=cl(V*V-v[i]*v[i],2*a[i])+d[i];
                r=L;
            }
        }
        else{
            if(v[i]<=V) continue;
            else{
                l=d[i];
                r=d[i]+fl(v[i]*v[i]-V*V,-2*a[i]);
            }
        }
        l=max(l,0);
        r=min(r,L);
        if(l>r) continue;
        int L=0,R=m+1,mid,resl,resr;
        while(L<R){
            mid=(L+R)>>1;
            if(p[mid]<l) L=mid+1;
            else R=mid;
        }
        resl=L;
        L=0,R=m+1;
        while(L<R){
            mid=(L+R+1)>>1;
            if(p[mid]>r) R=mid-1;
            else L=mid;
        }
```

```
        resr=L;
        if(resl>resr) continue;
        c[++tot]=(car){resl,resr};
    }
    sort(c+1,c+1+tot,[](car x,car y){return x.r!=y.r?x.r<y.r:x.l<y.l;});
    int ans=0;
    for(int i=1,lst=0;i<=tot;i++){
        if(c[i].l>lst) lst=c[i].r,ans++;
    }
    cout<<tot<<' '<<m-ans<<'\n';
}
signed main(){
    ios::sync_with_stdio(false),cin.tie(0),cout.tie(0);
    int T;cin>>T;
    while(T--) solve();
    return 0;
}
```

[CSP-S 2024] 染色

基本思路

算法一

考虑 DP，设 $dp_{i,j,k}$ 表示当前考虑到第 i 个数，编号最大的红色的编号为 j，编号最大的蓝色的编号为 k 时的最大得分。

转移方程略。

时间复杂度 $\Theta(n^3)$，预计得分 35 分。

算法二

发现 j,k 中必有一个为 i，可以略去。

于是设 $dp_{i,j}$ 表示当前考虑到第 i 个数，编号最大的与 i 异色的编号为 j 时的最大得分。

转移方程：

- i 与 $i-1$ 同色，$dp_{i,j}=dp_{i-1,j}+[a_i=a_{i-1}]a_i(j\le i-2)$。
- i 与 $i-1$ 异色，$dp_{i,i-1}=\max_{j=1}^{i-2}(dp_{i-1,j}+[a_i=a_j]a_i)$。

时间复杂度 $\Theta(n^2)$，预计得分 50 分。

算法三

考虑到算法二中第二条转移方程与值有关，转移不方便，将 DP 第二维改为值域。

于是设 $dp_{i,j}$ 表示当前考虑到第 i 个数，编号最大的与 i 异色的值为 j 时的最大得分。

转移方程（$V=max_{i=1}^{n}a_i$）：

- i 与 $i-1$ 同色，$dp_{i,j}=dp_{i-1,j}+[a_i=a_{i-1}]a_i(j\neq a_{i-1})$。
- i 与 $i-1$ 异色，$dp_{i,a_{i-1}}=\max\{dp_{i-1,a_i}+a_i,\max_{j=1}^{V}dp_{i-1,j}\}$。

预处理 $maxx_i=\max_{j=1}^{V}dp_{i,j}$。

时间复杂度 $\Theta(nV)$，预计得分 65 分。

算法四

考虑我们在 DP 转移时的操作：

设当前考虑到第 I 个数。

令 $f_i \leftarrow dp_{I-1,i}(i=1,2,\cdots,V), tmp \leftarrow f_{a_i}$。

若 $a_i = a_{i-1}$，$f_i \leftarrow f_i + a_i (i=1,2,\cdots,V)$。

$f_{a_{i-1}} \leftarrow \max\{tmp + a_i, \max_{j=1}^{V} f_j\}$.

只需实现一个支持全局加，全局查询最大值，单点赋值的数据结构，用一个 tag 记录目前的全局增量即可。

时间复杂度 $\Theta(n)$，预计得分 100 分。

代码实现

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const int N=200100,M=1000100;
const ll inf=0x3fffffffffffffff;
int n,a[N],V;
ll f[M],add,maxx,ans;
void solve(){
    cin>>n;
    for(int i=1;i<=n;i++) cin>>a[i],V=max(V,a[i]);
    for(int i=1;i<=V;i++) f[i]=-inf;
    add=0,maxx=0;
    for(int i=1;i<=n;i++){
        ll tmp=max(maxx,f[a[i]]+add+a[i]);
        if(a[i]==a[i-1]) add+=a[i],maxx+=a[i];
        f[a[i-1]]=max(f[a[i-1]],tmp-add);
        maxx=max(maxx,f[a[i-1]]+add);
    }
    ans=0;
    for(int i=1;i<=V;i++) ans=max(ans,f[i]+add);
    cout<<ans<<'\n';
}
signed main(){
    ios::sync_with_stdio(false),cin.tie(0),cout.tie(0);
    int T;cin>>T;
    while(T--) solve();
    return 0;
}
```

[CSP-S 2024] 擂台游戏

基本思路

本题考虑直接维护这个完全二叉树。

算法一

对于每个询问的每个人判断是否能够获胜：

- 在当前节点时 i 是擂主，直接按题意判断即可（补充的人能力设为 $+\infty$）。
- 在当前节点时 i 不是擂主
  - 若对方无可能获胜的补充选手，直接判断
  - 若对方有可能获胜的补充选手，则 i 可能在此获胜。

时间复杂度 $\Theta(Tnm\log n)$，预计得分 40 分。

算法二

发现每个节点可能获胜的人为：一个确定的选手和若干个补充的选手。

可以对每个节点维护其子树中全为确定的和全为补充的时的可能获胜的人。

用类似与线段树的方式对每个询问进行查询。

时间复杂度 $\Theta(T(n+m\log n))$，预计得分 60 分。

算法三

我们发现每次询问会重复访问一些信息，考虑优化。

定义时刻 i 表示第 $1,2,\cdots,i$ 的能力值确定时的答案。

容易发现，一个人在能力值确定时可能获胜的时刻为一区间（在某一时刻被打败后不可能再胜利）。

先处理出每个节点第一次有确定的胜者的时刻，以及该胜者。

计算答案时，直接记录几个限制交成的时刻区间即可。

时间复杂度 $\Theta(T(n+m))$，预计得分 100 分。

代码实现

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const int N=200100;
int n,m,pos[N<<1],sum[N<<1],a[N],op[N],A[N],c[N],K,X[4];
ll res[N];
char tmp[N];
#define ls id<<1
#define rs id<<1|1
void dfs1(int id,int dep){
    if(!dep){
        pos[id]=a[id-(1<<K)+1];
        sum[id]=id-(1<<K)+1;
        return;
    }
    dfs1(ls,dep-1),dfs1(rs,dep-1);
    if(!op[id]&&pos[ls]>=dep) sum[id]=sum[ls];
    else sum[id]=sum[rs];
    pos[id]=pos[ls|(op[id]^(pos[(ls)|op[id]]<dep))];
}
void dfs2(int id,int d,int dep,int L,int R){
    if(L>R) return;
    if(!dep){
        int tmp=id-(1<<K)+1;
        if(L<min(R,tmp)) res[L]+=tmp,res[min(R,tmp)]-=tmp;
        if(a[tmp]>=d&&max(tmp,L)<R)res[max(tmp,L)]+=tmp,res[R]-=tmp;
```

```
        return;
    }
    if(!op[id]){
        dfs2(ls,max(d,dep),dep-1,L,R);
        dfs2(rs,d,dep-1,L,pos[ls]>=dep?min(R,sum[ls]):R);
    }
    else{
        dfs2(ls,d,dep-1,L,pos[rs]>=dep?min(R,sum[rs]):R);
        dfs2(rs,max(d,dep),dep-1,L,R);
    }
}
#undef ls
#undef rs
void solve(){
    scanf("%d%d%d%d",&X[0],&X[1],&X[2],&X[3]);
    for(int i=1;i<=n;i++) a[i]=A[i]^X[i&3];
    dfs1(1,K);
    for(int i=0;i<=n;i++) res[i]=0;
    dfs2(1<<K,0,0,1,2);
    for(int k=1;k<=K;k++) dfs2(1<<(K-k),0,k,(1<<(k-1))+1,(1<<k)+1);
    for(int i=1;i<=n;i++) res[i]+=res[i-1];
    ll ans=0;
    for(int i=1;i<=m;i++) ans^=1ll*i*res[c[i]];
    printf("%lld\n",ans);
}
int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++) scanf("%d",&A[i]);
    for(int i=1;i<=m;i++) scanf("%d",&c[i]);
    K=__lg(n-1)+1;
    for(int k=K-1;k>=0;k--) for(int i=(1<<k);i<(1<<(k+1));i++)
scanf("%1d",&op[i]);
    int T;scanf("%d",&T);
    while(T--) solve();
    return 0;
}
```