比赛链接

补题情况

题目	蒋贤蒙	王赵安	王智彪
A	2	1	0
B	1	2	2
C	1	1	2
D	0	0	0
E	2	0	2
F	2	0	1
G	2	1	0
H	0	0	0
I	2	0	1
J	2	0	1

题解

A. Guess and lies

题意

一开始有一个数 $x\in [1,n]$，只有 A 知道，B 不知道。

B 每次可以给定一个 y，询问 $y\le x$ 是否成立，A 至多可以说一次谎，A 想要最大化询问次数，B 想要最小化询问次数。

假定 B 第一次问的数为 i 且 A 回答 $i\le x$ 成立，问还需要询问多少次。

题解

假定 B 询问的数为 y，如果 A 回答 $i\le x$ 成立，则将 $[1,y)$ 的说谎次数 $+1$，否则将 $[y,n]$ 的说谎次数 $+1$。

易知最后只剩下一个说谎次数不超过 1 的数就是答案。于是 B 相当于每个把 $[1,n]$ 分成两段，而 A 需要选择一段 $+1$。

不妨在每次操作后删去大于 1 的值，于是不难发现剩下的序列一定满足 11100011 的形式。

假定当前序列开头有连续 a 个 1，中间有连续 b 个 0，最后有连续 c 个 1。设 $\text{dp}(a,b,c)$ 表示当前局面剩下的操作次数。

不难发现可以枚举分割的位置，然后计算答案，时间复杂度 $O\left(n^4\right)$。

另外还有一个性质，设 $f(a,b,c,k)$ 表示当前局面如果决策位置为 k 则接下来还需要操作的次数，不难发现 $f(a,b,c,k)$ 为单峰函数。

同时，设当前局面的最优决策位置为 $p(a,b,c)$，有 $c_1\le c_2\to p(a,b,c_1)\le p(a,b,c_2)$。

于是考虑决策单调性，可以做到 $O\left(n^3\right)$，但还是不能通过本题数据范围。

```
const int MAXN=505;
int dp[MAXN][MAXN][MAXN];
int pos[MAXN][MAXN][MAXN];
int cal(int a,int b,int c,int p){
    if(p<=a)
    return max(dp[a-p][b][c],dp[p+b][0][0]);
    else if(p<=a+b){
        int l=p-a,r=a+b-p;
        return max(dp[a][l][r],dp[l][r][c]);
    }
    else{
        int l=p-a-b,r=a+b+c-p;
        return max(dp[a][b][l],dp[b+r][0][0]);
    }
}
int main(){
    int n=read_int();
    pos[1][0][0]=pos[0][1][0]=pos[0][0][1]=1;
    _rep(s,2,n){
        for(int a=s;a>=0;a--)_rep(b,0,s-a){
            int c=s-a-b;
            int &p=pos[a][b][c];
            if(c==0)
            p=1;
            else
            p=pos[a][b][c-1];
            while(p<s-1){
                if(cal(a,b,c,p)>=cal(a,b,c,p+1))
                p++;
                else
                break;
            }
            dp[a][b][c]=cal(a,b,c,p)+1;
        }
    }
    _for(i,0,n)
    space(dp[i][n-i][0]);
    return 0;
}
```

不难发现 B 询问的次数最多 $O\left(\log n\right)$ 次，于是交换 dp 的值域和第三维。

设 $\text{dp}(i,a,b)$ 表示 i 次操作能处理的局面 (a,b,c) 中 c 的最大值，如果 $\text{dp}(i,a,b)=-1$ 表示 i 次操作不能处理局面 $(a,b,0)$。

关于状态转移也比较复杂，要分三种情况。

如果 $k\in [1,a]$ 则 $(a,b,c)\to (k+b,0,0),(a-k,b,c)$ 此时要保证 $\text{dp}(i-1,k+b,0)\ge 0$ 才算合法，否则 B 就不能 i 步走完。

合法情况下有 $\text{dp}(i,a,b)\gets \text{dp}(i-1,a-k,b)$ 不难发现 $\text{dp}(i-1,a-k,b)$ 随 k 增加递减，于是取合法情况的分界点即可。

如果 $k\in (a,a+b]$ 则 $(a,b,c)\to (a,k-a,a+b-k),(k-a,a+b-k,c)$ 此时要保证 $\text{dp}(i-1,a,k-a)\ge a+b-k$ 才算合法。

合法情况下有 $\text{dp}(i,a,b)\gets \text{dp}(i-1,k-a,a+b-k)$ 同样 $\text{dp}(i-1,k-a,a+b-k)$ 也随 k 增加递减。

最后 $k\in (a+b,a+b+c]$ 不妨记 $k'=k-a-b$ 则 $(a,b,c)\to (b,0,c-k'),(a,b,k')$

此时要保证 $\text{dp}(i-1,b,0),\text{dp}(i-1,a,b)\ge 0$ 才算合法，不难发现，此时 $c=c-k'+k'=\text{dp}(i-1,b,0)+\text{dp}(i-1,a,b)$

同时 $p(i,a,b)$ 表示当前状态最优决策位置，关于 a 貌似具有不严格单峰性，根据打表结果有 $p(i,a,b)\ge p(i,a-1,b)-1$

于是可以 $O(n^2\log n)$ 通过此题。

```
const int MAXV=20,MAXN=2005;
int dp[MAXV][MAXN][MAXN];
bool check(int i,int a,int b,int p){
    if(p<=a)
    return dp[i-1][p+b][0]>=0;
    else
    return dp[i-1][a][p-a]>=a+b-p;
}
int main(){
    int n=read_int();
    mem(dp,-1);
    dp[0][0][0]=1;
    dp[0][1][0]=dp[0][0][1]=0;
    _for(i,1,MAXV){
        _rep(b,0,n)for(int a=0,p=0;a<=n-b;a++){
            while(p<a+b&&check(i,a,b,p))p++;
            while(p&&!check(i,a,b,p))p--;
            dp[i][a][b]=(p<=a)?dp[i-1][a-p][b]:dp[i-1][p-a][a+b-p];
            if(dp[i-1][a][b]>=0&&dp[i-1][b][0]>=0)
            dp[i][a][b]=max(dp[i][a][b],dp[i-1][a][b]+dp[i-1][b][0]);
        }
    }
    _for(i,0,n){
        _for(j,0,MAXV){
            if(dp[j][i][n-i]>=0){
                space(j);
                break;
            }
```

```
            }
        }
        return 0;
}
```

G. Yu Ling(Ling YueZheng) and Colorful Tree

题意

给定一棵点权树，初始值各点权值为 0，接下来两种操作：

1. $w(u)\gets x$保证 $w(u)$ 之前一定为 0，该类操作的 x 都不相同且 $1\le x\le n$
2. 给定 $1\le l,r,x\le n$查询 u 的最近祖先节点 v满足 $x\mid w(v),l\le w(v)\le r$输出 u,v 之间的距离

题解

考虑离线操作，对每个权值，维护对他有贡献的操作序列。

首先对操作 1，显然 x 会对所有是 x 的因子的权值产生贡献，于是将操作 1 加入到所有 x 的因子的操作序列中。

对操作 2，直接丢到权值 x 的操作序列处理即可。

接下来单独考虑每个权值的操作序列，发现操作序列一定都满足 $x\mid w(v)$ 这个约束。

于是只需要考虑 $l\le w(v)\le r$ 和 v 是 u 的最近祖先这两个约束。

首先转化一下v 是 u 的最近祖先这个约束，不难发现满足该约束的 v 就是 u 的祖先中 dfs 序最大的点。

维护一个二维矩阵 $C$$C[x][y]$ 表示 dfs 序最大的 v满足 $w(v)=x$ 且 $w(v)$ 是节点 y 的祖先。

于是对操作 1 等价于修改

$$ C[x][\text{dfn}(u)\sim \text{dfn}(u)+\text{sz}(u)-1]\gets \text{dfn}(u) $$

操作 2 等价于查询

$$ \max\left(C[l\sim r][\text{dfn}(u)]\right) $$

考虑树套树维护 C考虑修改操作共 $O(n\log n)$ 次，查询操作 $O(n)$于是总时间复杂度 $O\left(n\log^3 n\right)$

关于空间复杂度，首先对每个权值，由于他的倍数最多只有 $O(n)$ 个，所以修改操作最大只有 $O(n)$ 次。

树套树第二维是动态开点线段树，于是空间复杂度 $O\left(n\log^2 n\right)$注意处理完每个权值的操作序列后要删除线段树，否则会爆空间。

```
const int MAXN=1.2e5+5,MAXM=150;
struct Edge{
    int to,w,next;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v,int w){
    edge[++edge_cnt]=Edge{v,w,head[u]};
    head[u]=edge_cnt;
}
int dfl[MAXN],dfr[MAXN],dfu[MAXN],dfs_t;
LL dis[MAXN];
void dfs(int u,int fa){
    dfl[u]=++dfs_t;
    dfu[dfs_t]=u;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)continue;
        dis[v]=dis[u]+edge[i].w;
        dfs(v,u);
    }
    dfr[u]=dfs_t;
}
struct node{
    int u,x,l,r,id;
    node(int u=0,int x=0,int l=0,int r=0,int id=0){
        this->u=u;
        this->x=x;
        this->l=l;
        this->r=r;
        this->id=id;
    }
};
namespace Tree{
    int ch[MAXN*MAXM][2],s[MAXN*MAXM],cnt;
    void clear(){cnt=0;}
    void update(int &k,int vl,int vr,int ql,int qr,int v){
        if(!k){
            k=++cnt;
            s[k]=0;
            ch[k][0]=ch[k][1]=0;
        }
        if(ql<=vl&&vr<=qr){
            s[k]=max(s[k],v);
            return;
        }
        int vm=vl+vr>>1;
        if(vm>=ql)
        update(ch[k][0],vl,vm,ql,qr,v);
        if(vm<qr)
        update(ch[k][1],vm+1,vr,ql,qr,v);
```

```
    }
    int query(int k,int vl,int vr,int pos){
        if(!k)return 0;
        if(vl==vr)return s[k];
        int vm=vl+vr>>1;
        if(vm>=pos)
        return max(s[k],query(ch[k][0],vl,vm,pos));
        else
        return max(s[k],query(ch[k][1],vm+1,vr,pos));
    }
}
int rt0[MAXN],rt[MAXN],n;
#define lowbit(x) (x&(-x))
void update(int pos,int u){
    Tree::update(rt0[pos],1,n,dfl[u],dfr[u],dfl[u]);
    while(pos<=n){
        Tree::update(rt[pos],1,n,dfl[u],dfr[u],dfl[u]);
        pos+=lowbit(pos);
    }
}
int query(int l,int r,int u){
    int ans=0;
    while(l<=r){
        ans=max(ans,Tree::query(rt0[r],1,n,dfl[u]));
        for(--r;r-l>=lowbit(r);r-=lowbit(r))
        ans=max(ans,Tree::query(rt[r],1,n,dfl[u]));
    }
    return ans;
}
void del(int pos){
    rt0[pos]=0;
    while(pos<=n){
        rt[pos]=0;
        pos+=lowbit(pos);
    }
}
vector<int> d[MAXN];
vector<node> opt[MAXN];
LL ans[MAXN];
void solve(int v){
    for(node q:opt[v]){
        if(q.x==0){
            int t=query(q.l,q.r,q.u);
            if(t==0)
            ans[q.id]=-1;
            else
            ans[q.id]=dis[q.u]-dis[dfu[t]];
        }
        else
        update(q.x,q.u);
```

```
    }
    for(node q:opt[v]){
        if(q.x!=0)
        del(q.x);
    }
    Tree::clear();
}
int main(){
    n=read_int();
    int q=read_int();
    _for(i,1,n){
        int u=read_int(),v=read_int(),w=read_int();
        Insert(u,v,w);
        Insert(v,u,w);
    }
    dfs(1,0);
    _rep(i,1,n){
        for(int j=i;j<=n;j+=i)
        d[j].push_back(i);
    }
    int qcnt=0;
    while(q--){
        int type=read_int(),u=read_int();
        if(type==0){
            int x=read_int();
            for(int v:d[x])
            opt[v].push_back(node(u,x));
        }
        else{
            int l=read_int(),r=read_int(),x=read_int();
            opt[x].push_back(node(u,0,l,r,qcnt++));
        }
    }
    _rep(i,1,n)
    solve(i);
    _for(i,0,qcnt){
        if(ans[i]==-1)
        puts("Impossible!");
        else
        enter(ans[i]);
    }
    return 0;
}
```

I. Kuriyama Mirai and Exclusive Or

题意

给定一个序列 A□接下来两种操作：

1. $a_i\gets a_i\oplus x(i\in [l,r])$
2. $a_i\gets a_i\oplus (x+i-l)(i\in [l,r])$

题解

$$ a_n=\left(\bigoplus_{i=1}^n b_i\right)\oplus\left(\bigoplus_{i=1}^n\bigoplus_{k=0}^{29} 2^k c(i,k)\right)\\ c(n,k)=\bigoplus_{i\times 2^k\le n} d(n-i\times 2^k,k) $$

简单来讲 a_n 是 $b_i,c(i,k)$ 的前缀和，$c(n,k)$ 是 $d(i,k)$ 的隔 2^k 项前缀和。

然后操作 1 只需要修改 b_i 即可。操作 2 的影响按位考虑影响，将 $[l,r]$ 区间操作等效于 $[l,\inf),[r+1,\inf)$ 两次操作。

每次操作对每个位是按 0011110000111110000 的规律周期性变化的。

对该序列进行一次差分，于是得到 0010001000100010000，这个可以利用 $c(n,k)$ 维护。

再一次差分得到 0010000000000000000 发现可以利用 $d(n,k)$ 维护。

最后处理一下最开始非周期的部分即可，时间复杂度 $O((n+q)\log v)$。

```
const int MAXN=6e5+5,MAXB=30,mod=1<<30;
int n,a[MAXN],b[MAXN];
bitset<MAXN> c[MAXB];
void update(int pos,int v){
    _for(i,0,MAXB){
        int cyc=1<<(i+1),p1=pos%cyc,p2=((cyc-
v%cyc)%cyc+(1<<i))%cyc,d=pos>>(i+1);
        if(p1>=p2){
            if(p1<p2+(1<<i)){
                b[pos]^=1<<i;
                if(d*cyc+p2+(1<<i)<MAXN)
                b[d*cyc+p2+(1<<i)]^=1<<i;
            }
            d++;
        }
        else if(p1<p2-(1<<i)){
            b[pos]^=1<<i;
            if(d*cyc+p2-(1<<i)<MAXN)
            b[d*cyc+p2-(1<<i)]^=1<<i;
        }
        if(d*cyc+p2<MAXN)
        c[i].flip(d*cyc+p2);
    }
}
int main() {
    n=read_int();
    int q=read_int();
    _for(i,0,n)a[i]=read_int();
    while(q--){
```

```
        int opt=read_int(),l=read_int()-1,r=read_int()-1,x=read_int();
        if(opt==0){
            b[l]^=x;
            b[r+1]^=x;
        }
        else{
            update(l,(x-l+mod)%mod);
            update(r+1,(x-l+mod)%mod);
        }
    }
    _for(i,0,MAXB){
        _for(j,0,n)if(j+(1<<i)<MAXN)
        c[i][j+(1<<i)]=c[i][j+(1<<i)]^c[i][j];
        _for(j,1,n)
        c[i][j]=c[i][j]^c[i][j-1];
        _for(j,0,n)
        a[j]^=(c[i][j]<<i);
    }
    _for(i,1,n)b[i]^=b[i-1];
    _for(i,0,n)a[i]^=b[i];
    _for(i,0,n)
    space(a[i]);
    return 0;
}
```

赛后总结

jxm找操作中的不变量可能是解题的一种思路，比如这场的 B 或者博弈论题 A 据说是 dp 优化题，但没看懂题解，先咕了。

update补完A了，感觉性质比较玄学。