Codeforces Global Round 13

比赛链接

F. Magnets

题意

给定 n 个磁体，编号 $1\sim n$磁体有三种类型 $S,N,-$其中 $-$ 代表无磁性。允许 $n+\lfloor \log n\rfloor$ 次询问。

每次询问选择若干磁体，分成两组，然后用装置测量两组磁体之间的受力。

其中，设第 i 组有 s_i 个 S 型磁体n_i 个 N 型磁体，则受力为 $s_1s_2+n_1n_2-s_1n_2-s_2n_1$若受力超过 n 则测量装置损坏。

要求在不损坏装置的前提下在允许询问次数内找到所有无磁性磁体。

数据保证至少有两个有磁性的磁体和一个无磁性的磁体。

题解

每次询问 $1\sim i-1$ 和 i 号磁体两组之间的受力，当受力不为 0 时 i 号磁体一定有磁性$1\sim i-1$ 中一定恰有一个有磁性磁体。

考虑二分找到 $1\sim i-1$ 中的有磁性磁体，然后用第 i 号磁体检验 $i+1\sim n$ 磁体的磁性。查询次数 $n-1+\lceil \log n\rceil$

```
int query(int a,int b){
    puts("? 1 1");
    enter(a);
    enter(b);
    fflush(stdout);
    int t;scanf("%d",&t);
    return t;
}
int query(int pos,int lef,int rig){
    printf("? 1 %d\n",rig-lef+1);
    enter(pos);
    _rep(i,lef,rig)space(i);
    puts("");
    fflush(stdout);
    int t;scanf("%d",&t);
    return t;
}
int solve(int pos,int lef,int rig){
```

```
    int mid=lef+rig>>1;
    if(lef==rig)return mid;
    int t=query(pos,lef,mid);
    if(t)return solve(pos,lef,mid);
    else
    return solve(pos,mid+1,rig);
}
int main()
{
    int T=read_int();
    while(T--){
        int n;scanf("%d",&n);
        vector<int> ans;
        int pos=2;
        while((query(pos,1,pos-1))==0)pos++;
        int t=solve(pos,1,pos-1);
        _for(i,1,pos){
            if(i!=t)
            ans.push_back(i);
        }
        _rep(i,pos+1,n){
            if(query(pos,i)==0)
            ans.push_back(i);
        }
        printf("! %d ",ans.size());
        _for(i,0,ans.size())space(ans[i]);
        puts("");
        fflush(stdout);
    }
    return 0;
}
```

G. Switch and Flip

题意

给定 $n(n\ge 3)$ 枚硬币，初始时 i 号硬币位于位置 a_i 且正面朝上。每次操作可以交换两枚硬币位置同时将两枚硬币翻面。

要求给出方案在至多 $n+1$ 次操作内将所有硬币归位(i 号硬币位于位置 i 且正面朝上)。

题解

首先发现当循环 S 中有两枚硬币反面朝上时，对一个反面朝上硬币，直接与他标号位置做交换直到遇到另一枚反面朝上硬币。

于是可以经过 $|S|-2$ 次操作将除这两个位置以外的其他位置归位，然后交换这两个位置即可。

对两个循环 S_1,S_2 直接交换这两个循环中的任意一枚硬币，即可得到上述循环，于是可以 $|S_1|+|S_2|$ 次完成归位。

如果最后剩下一个循环，当这个循环大小低于 n 时，直接将一个已经归位的位置当作长度为 1 的循环进行归位即可。

如果这个循环大小为 n 则该循环至少有 3 个元素，不妨设循环中连续的三个位置为 p_1,p_2,p_3

依次交换位置 $(p_1,p_2),(p_2,p_3)$ 于是可以得到位置 p_1,p_2 反面朝上的循环。

于是可以至多 $n+1$ 次操作完成所有归位。

```
const int MAXN=2e5+5;
int a[MAXN];
bool vis[MAXN];
vector<pair<int,int> > ans;
void Swap(int p1,int p2){
    swap(a[p1],a[p2]);
    ans.push_back(make_pair(p1,p2));
}
void solve(int u,int v){
    while(a[u]!=v)Swap(u,a[u]);
    while(a[v]!=u)Swap(v,a[v]);
    Swap(u,v);
}
int main()
{
    int n=read_int(),last=0;
    _rep(i,1,n)a[i]=read_int();
    _rep(i,1,n){
        if(vis[i])continue;
        vis[i]=true;
        if(a[i]==i)continue;
        int pos=i;
        while(a[pos]!=i){
            pos=a[pos];
            vis[pos]=true;
        }
        if(last==0)
        last=i;
        else{
            Swap(i,last);
            solve(i,last);
            last=0;
        }
    }
    if(last){
        bool flag=false;
        _rep(i,1,n){
            if(a[i]==i){
                Swap(i,last);
```

```
                solve(i,last);
                flag=true;
                break;
            }
        }
        if(!flag){
            int p1=last,p2=a[p1],p3=a[p2];
            Swap(p1,p2);
            Swap(p2,p3);
            solve(p1,p2);
        }
    }
    enter(ans.size());
    _for(i,0,ans.size())
    printf("%d %d\n",ans[i].first,ans[i].second);
    return 0;
}
```

H. Yuezheng Ling and Dynamic Tree

题意

给定 $a_2\sim a_n$ 表示结点 i 的父结点，其中 $a_i\lt i$。接下来两种操作：

1. $a_i\to \max(a_i-x,1),i\in [l,r]$
2. 查询 u,v 的 LCA

题解

考虑分块，块大小为 $O(\sqrt n)$。对结点 i 用 p_i 维护 i 在同一个块中的最远祖先结点。

于是，对于查询操作，可以类似树剖查询做到 $O(\sqrt n)$。

对于修改操作，易知 a_i 最多减小 $O(\sqrt n)$ 次后 p_i 固定，因为此时必有 a_i 和 i 不在同一个块，于是 $p_i=i$。

当一个块满足所有 $p_i=i$ 时对整块修改操作只需要用懒标记维护 a_i 即可。

每个块最多进行 $O(\sqrt n)$ 次整块修改操作，每次整块修改操作复杂度 $O(\sqrt n)$，共 $O(\sqrt n)$ 个块，于是整块修改操作的复杂度为 $O(n\sqrt n)$。

对于一个询问，最多有两个部分块修改操作，暴力维护即可，时间复杂度 $O(q\sqrt n)$。

总时间复杂度 $O\left((n+q)\sqrt n\right)$。

```
const int MAXN=1e5+5,MAXB=sqrt(MAXN)+5;
int a[MAXN],p[MAXN],blk_id[MAXN],blk_sz;
```

```
struct Block{
    int L,R,lazy,lazy_en;
    void update_p(){
        lazy_en=true;
        _rep(i,L,R){
            if(blk_id[i]==blk_id[a[i]]){
                lazy_en=false;
                p[i]=p[a[i]];
            }
            else
            p[i]=i;
        }
    }
    void update_a(int l,int r,int x){
        _rep(i,l,r)
        a[i]=(i==1)?0:max(a[i]-x,1);
        update_p();
    }
    void update_a(int x){
        if(lazy_en)lazy+=x;
        else
        update_a(L,R,x);
    }
}blk[MAXB];
void update(int l,int r,int x){
    int pos1=blk_id[l],pos2=blk_id[r];
    if(pos1==pos2)
    blk[pos1].update_a(l,r,x);
    else{
        blk[pos1].update_a(l,blk[pos1].R,x);
        _for(i,pos1+1,pos2)
        blk[i].update_a(x);
        blk[pos2].update_a(blk[pos2].L,r,x);
    }
}
int A(int pos){
    return pos==1?0:max(a[pos]-blk[blk_id[pos]].lazy,1);
}
int query(int u,int v){
    while(p[u]!=p[v]){
        if(A(p[u])<A(p[v]))swap(u,v);
        u=A(p[u]);
    }
    while(u!=v){
        if(u<v)swap(u,v);
        u=A(u);
    }
    return u;
}
int main()
{
```

```
    int n=read_int(),q=read_int();
    _rep(i,2,n)a[i]=read_int();
    blk_sz=sqrt(n)+1;
    for(int l=1,r=0,blk_cnt=1;r!=n;l=r+1,blk_cnt++){
        r=min(l+blk_sz-1,n);
        blk[blk_cnt].L=l;
        blk[blk_cnt].R=r;
        _rep(i,l,r)
        blk_id[i]=blk_cnt;
        blk[blk_cnt].update_p();
    }
    while(q--){
        int t=read_int();
        if(t==1){
            int l=read_int(),r=read_int(),x=read_int();
            update(l,r,x);
        }
        else{
            int u=read_int(),v=read_int();
            enter(query(u,v));
        }
    }
    return 0;
}
```