# 错题集 1

## 1□Fake Maxpooling

链接

### 题意

给定一个 $n\times m$ 矩阵□$a_{i,j}=\text{lcm}(i,j)$□设 $f(i,j)=\{\max a_{x,y}|i\le x\lt i+k,j\le y\lt j+k\}$□求

\begin{equation}\sum_{i=1}^{n-k+1}\sum_{j=1}^{m-k+1}f(i,j)\end{equation}

### 题解

记忆化搜索求 $\text{gcd}$□两次单调队列维护最大值。

第一次单调队列维护每行长度为 $k$ 的区间的最大值，第二次单调队利用 $k$ 列每行长度为 $k$ 的区间的最大值。

时间复杂度 $O(nm)$□

```
const int MAXN=5005;
int a[MAXN][MAXN],b[MAXN][MAXN],que[MAXN];
int main()
{
    int n=read_int(),m=read_int(),k=read_int(),u,v;
    _rep(i,1,n)
    _rep(j,1,m){
        if(!a[i][j]){
            for(int k=1;k*i<=n&&k*j<=m;k++)
            a[i*k][j*k]=i*j*k;
        }
    }
    _rep(i,1,n){
        int front=1,tail=0;
        _for(j,1,k){
            while(front<=tail&&a[i][que[front]]<=a[i][j])front++;
            que[++tail]=j;
        }
        _rep(j,k,m){
            while(front<=tail&&j-que[front]>=k)front++;
            while(front<=tail&&a[i][que[front]]<=a[i][j])front++;
            que[++tail]=j;
            b[i][j]=a[i][que[front]];
        }
```

```
    }
    LL sum=0;
    _rep(j,k,m){
        int front=1,tail=0;
        _for(i,1,k){
            while(front<=tail&&b[que[front]][j]<=b[i][j])front++;
            que[++tail]=i;
        }
        _rep(i,k,n){
            while(front<=tail&&i-que[front]>=k)front++;
            while(front<=tail&&b[que[front]][j]<=b[i][j])front++;
            que[++tail]=i;
            sum+=b[que[front]][j];
        }
    }
    enter(sum);
    return 0;
}
```

# 2□Columns Swaps

链接

## 题意

给定一个 $2\times n$ 的表格，定义一次操作为交换同一列的上下两个元素。

问至少要经过多少次操作才能使表格两行均为 $1\sim n$ 的全排列。(如果无解，输出 $-1$)

## 题解

首先如果某个数字在表格中出现次数不为 $2$ 必无解。

否则记录每个数的两次出现的行号和列号。

建图，每个列代表一个点。对图进行某种染色，若一个点染黑色代表该列要交换，若一个点染白色代表该列不要交换。

考虑每对相同数的行号。如果行号相同，则两个数所在列必须有一个交换，一个不交换，两列所代表点连一条黑边，代表这两点颜色不同。

如果行号不相同，则两个数所在列要么都交换，要么都不交换，两列所代表点连一条白边，代表这两点颜色相同。

对每个连通块进行染色，初始点颜色设为 $0$，与初始点不同的颜色设为 $1$。最后如果 $0$ 色点多则 $0$ 色为白色，否则 $1$ 色为白色。

由于染色过程遇到已标记点会跳过，所以最后需要在染色后枚举边集确定每条边的两个点颜色是否满足条

件。

```
const int MAXN=2e5+5;
struct Edge{
    int to,type,next;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v,int type){
    edge[++edge_cnt]=Edge{v,type,head[u]};
    head[u]=edge_cnt;
}
vector<int> r[MAXN],c[MAXN];
int block_id[MAXN],block_cnt,color[MAXN],color_cnt[MAXN][2];
void dfs(int u,int c){
    block_id[u]=block_cnt;color[u]=c;
    color_cnt[block_cnt][c]++;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(block_id[v])
        continue;
        dfs(v,c^edge[i].type);
    }
}
int main()
{
    int t=read_int();
    while(t--){
        mem(block_id,0);block_cnt=0;
        mem(head,0);edge_cnt=0;
        mem(color_cnt,0);
        int n=read_int(),a;
        _rep(i,1,n)
        r[i].clear(),c[i].clear();
        _rep(i,1,n){
            a=read_int();
            c[a].push_back(i);
            r[a].push_back(1);
        }
        _rep(i,1,n){
            a=read_int();
            c[a].push_back(i);
            r[a].push_back(2);
        }
        bool flag=true;
        _rep(i,1,n){
            if(c[i].size()>2){
                puts("-1");
                flag=false;
                break;
            }
```

```
        }
        if(!flag)
        continue;
        _rep(i,1,n){
            Insert(c[i][0],c[i][1],r[i][0]==r[i][1]);
            Insert(c[i][1],c[i][0],r[i][0]==r[i][1]);
        }
        int ans=0;
        _rep(i,1,n){
            if(!block_id[i]){
                block_cnt++;
                dfs(i,0);
                ans+=min(color_cnt[block_cnt][0],color_cnt[block_cnt][1]);
            }
        }
        for(int i=1;i<edge_cnt;i+=2){
            if(color[edge[i].to]^color[edge[i+1].to]^edge[i].type){
                puts("-1");
                flag=false;
                break;
            }
        }
        if(!flag)
        continue;
        enter(ans);
        _rep(i,1,n){
if(color[i]^(color_cnt[block_id[i]][0]<color_cnt[block_id[i]][1]))
            space(i);
        }
        puts("");
    }
    return 0;
}
```

# 3□Columns Swaps

链接

## 题意

给定一个 $1\sim n$ 的数列，保证 $n$ 为偶数。对这个数列经行两两配对，操作费用为每组配对数字之差的绝对值的总和。

要求进行两次上述操作，操作费用和最小，且不存在某个配对同时存在在两次操作中。

### 题解

将每个数字当成一个节点，两次配对后将出现若干个偶环。显然偶环长度只能为 $4$ 或 $6$，且偶环取相邻元素最优。

直接 $\text{dp}$ 即可。(比赛时误以为长度为 $6$ 的环至多只有一个了，唉)

```
const int MAXN=2e5+5;
int a[MAXN],dp[MAXN];
int main()
{
    int t=read_int();
    while(t--){
        int n=read_int();
        _rep(i,1,n)
        a[i]=read_int();
        sort(a+1,a+n+1);
        dp[4]=a[4]-a[1];
        dp[6]=a[6]-a[1];
        dp[8]=a[8]-a[5]+dp[4];
        for(int i=10;i<=n;i+=2)
        dp[i]=min(dp[i-4]+a[i]-a[i-3],dp[i-6]+a[i]-a[i-5]);
        enter(2*dp[n]);
    }
    return 0;
}
```

## 4□Just Shuffle

链接

### 题意

已知某个置换对初始排列 $1,2,\cdots n$ 连续迭代 $k$ 次的置换，求原置换(保证 $k$ 为大质数)。

### 题解

首先对任意一个置换的循环节，设其长度为 $m$□该循环节对初始排列连续迭代 $t$ 次等价与对初始排列连续迭代 $t\bmod m$ 次。

已知某个循环节是对初始排列连续迭代 $k$ 次的结果，将当前结果对初始排列连续迭代 $k^{-1}\bmod m$ 次。

这等效于将原置换对初始排列连续迭代 $k\ast k^{-1}\equiv 1\bmod m$ 次，即结果仍为原置换。

```
const int MAXN=1e5+5;
int a[MAXN],vis[MAXN],pos[MAXN],cyc_cnt;
vector<int> cyc[MAXN];
void dfs(int u){
    if(vis[u])
    return;
    vis[u]=true;
    cyc[cyc_cnt].push_back(u);
    dfs(a[u]);
}
LL exgcd(LL a,LL b,LL &tx,LL &ty){
    if(b==0){
        tx=1,ty=0;
        return a;
    }
    LL re=exgcd(b,a%b,ty,tx);
    ty-=a/b*tx;
    return re;
}
int main()
{
    int n=read_int(),k=read_int();
    _rep(i,1,n)
    a[i]=read_int();
    _rep(i,1,n){
        if(!vis[i]){
            dfs(i);
            cyc_cnt++;
        }
    }
    _for(i,0,cyc_cnt){
        int len=cyc[i].size();
        if(len==1){
            pos[cyc[i][0]]=cyc[i][0];
            continue;
        }
        LL inv,temp;
        exgcd(k%len,len,inv,temp);
        inv=(inv%len+len)%len;
        _for(j,0,cyc[i].size())
        pos[cyc[i][j]]=cyc[i][(j+inv)%len];
    }
    _rep(i,1,n)
    space(pos[i]);
    return 0;
}
```

# 5□Hard Gcd Problem

链接

## 题意

给定一个 $n$□要求找出最大的集合 $A,B$□使得 $A,B\subset \{1,2,\cdots n\}$□且 $|A|=|B|,A\cap B=\emptyset$□

同时设 $A=\{x_1,x_2,\cdots x_m\}$□$B=\{y_1,y_2,\cdots y_m\}$□有 $(x_i,y_i)\gt 1$□

## 题解

显然答案不大于 $\frac {n-1-|C|}2,C=\{p|p\gt \frac n2\}$□

现在构造满足该上界的解。

然后从 $3$ 开始枚举不属于 $C$ 的质因子 $p$□把之前未访问的质因子的倍数丢掉桶里，显然这个桶里一定会有 $2p$□因为 $2p$ 不可能在之前被访问。

如果桶里有偶数个数，直接两两配对，否则剩下一个 $2p$□

最后只剩下偶数，直接两两配对。

```
const int MAXP=2e5+20;
bool vis[MAXP],vis2[MAXP];
int prime[MAXP],cnt;
vector<int> kp[MAXP];
void Prime(){
    vis[1]=true;
    _for(i,2,MAXP){
        if(!vis[i])prime[cnt++]=i;
        for(int j=0;j<cnt&&i*prime[j]<MAXP;j++){
            vis[i*prime[j]]=true;
            if(i%prime[j]==0) break;
        }
    }
}
int main()
{
    Prime();
    int t=read_int(),n;
    while(t--){
        n=read_int();
        _rep(i,2,n)
        vis2[i]=0;
        for(int i=0;prime[i]*2<=n;i++)
        kp[i].clear();
```

```
        for(int i=1;prime[i]*2<=n;i++){
            vis2[prime[i]*2]=true;
            kp[i].push_back(prime[i]*2);
            for(int j=1;prime[i]*j<=n;j++){
                if(!vis2[prime[i]*j]){
                    vis2[prime[i]*j]=true;
                    kp[i].push_back(prime[i]*j);
                }
            }
        }
        int lost=0;
        _rep(i,1,n){
            if(!vis2[i]){
                if(i%2==0){
                    vis2[i]=true;
                    kp[0].push_back(i);
                }
                else
                lost++;
            }
        }
        enter((n-lost)/2);
        for(int i=1;prime[i]*2<=n;i++){
            if(kp[i].size()%2==0){
                for(int j=0;j<kp[i].size();j+=2)
                printf("%d %d\n",kp[i][j],kp[i][j+1]);
            }
            else{
                kp[0].push_back(kp[i][0]);
                for(int j=1;j<kp[i].size();j+=2)
                printf("%d %d\n",kp[i][j],kp[i][j+1]);
            }
        }
        for(int i=1;i<kp[0].size();i+=2)
        printf("%d %d\n",kp[0][i-1],kp[0][i]);
    }
    return 0;
}
```

# 6、Chess Strikes Back

链接

## 题意

一个 $2n\times 2m$ 的棋盘，对棋盘进行二染色，行列号和为偶数则为白格，否则为黑格。

已知国王攻击范围为 $3\times 3$要求在白格上放 $n\times m$ 个国王，且保证国王不相互攻击。

$q$ 次修改，每次询问先修个某个位置的白格状态。如果此时白格未被禁用，则将其禁用，否则将其恢复正常。

要求输出每次修改后是否存在可以合法放置所有国王的方案。

## 题解

把 $2n\times 2m$ 的棋盘划分成 $2\times 2$ 的正方形，显然每个正方形里必须放一个国王，且只有左上角和右下角可以放置国王。

如果正方形左上角被禁用，则即该正方形为 $L$ 型正方形。

如果正方形右下角被禁用，则即该正方形为 $R$ 型正方形。注意一个正方形可以既是 $L$ 型又是 $R$ 型。

显然如果存在 $L$ 型正方形，则 $L$ 型正方形右下方(包括正下方和正右方)的正方形国王必须放在右下角。

如果存在 $R$ 型正方形，则 $R$ 型正方形左上方(包括正上方和正左方)的正方形国王必须放在左上角。

所以当且仅当 $L$ 型正方形右下方出现 $R$ 型正方形无合法方案。

考虑用 $set$ 维护每行中 $L$ 型最靠左的位置(记为 $low$)和 $R$ 型正方形最靠右的位置(记为 $high$)

则合法方案存在当且仅当对任意 $j$有第 $j$ 行的 $high\lt$ 前 $j$ 行的 $low$ 的最小值。

考虑线段树维护即可，当前区间合法当且仅当左区间合法，右区间合法，左区间 $low$ 最小值大于 右区间 $high$ 最大值。

时间复杂度 $O\left((n+q)(\log n+\log q)\right)$

```
const int MAXN=2e5+5,Inf=1e9;
set<int> low[MAXN<<2];
set<int,greater<int> > high[MAXN<<2];
int lef[MAXN<<2],rig[MAXN<<2],v_low[MAXN<<2],v_high[MAXN<<2];
bool flag[MAXN<<2];
void push_up(int k){
    flag[k]=flag[k<<1]&flag[k<<1|1];
    flag[k]&=v_low[k<<1]>v_high[k<<1|1];
    v_low[k]=min(v_low[k<<1],v_low[k<<1|1]);
    v_high[k]=max(v_high[k<<1],v_high[k<<1|1]);
}
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R;
    if(L==R){
        flag[k]=true;
        low[k].insert(Inf);
        v_low[k]=Inf;
        high[k].insert(-Inf);
        v_high[k]=-Inf;
        return;
```

```
    }
    int M=L+R>>1;
    build(k<<1,L,M);
    build(k<<1|1,M+1,R);
    push_up(k);
}
void update(int k,int pos,int type,int v){
    if(lef[k]==rig[k]){
        if(type){
            if(low[k].find(v)==low[k].end())low[k].insert(v);
            else low[k].erase(v);
        }
        else{
            if(high[k].find(v)==high[k].end())high[k].insert(v);
            else high[k].erase(v);
        }
        v_low[k]=*low[k].begin();
        v_high[k]=*high[k].begin();
        flag[k]=v_low[k]>v_high[k];
        return;
    }
    int mid=lef[k]+rig[k]>>1;
    if(mid>=pos)
    update(k<<1,pos,type,v);
    else
    update(k<<1|1,pos,type,v);
    push_up(k);
}
int main()
{
    int n=read_int(),m=read_int(),q=read_int(),a,b;
    build(1,1,n);
    _for(i,0,q){
        int r=read_int(),c=read_int();
        update(1,(r+1)>>1,r&1,c);
        if(flag[1])
        puts("YES");
        else
        puts("NO");
    }
    return 0;
}
```

# 7、Classical String Problem

链接

### 题意

有一个字符串，有两种操作：

1. 询问第 $x$ 个字符
2. 把最左边的 $x$ 个字符搬到最右边或把最右边 $x$ 个字符搬到最左边

### 题解

如果把字符串当成一个环，发现操作 $2$ 变成了旋转操作，不改变环上字符串的相对位置。

所以只需要维护环首位置即可($\text{ps}$ 此题专卡无脑 $\text{Splay}$ 选手)

```
const int MAXN=2e6+5;
char buf[MAXN];
int main()
{
    int pos=0,q,x,len;char opt;
    scanf("%s",buf);
    q=read_int(),len=strlen(buf);
    while(q--){
        opt=get_char();x=read_int();
        if(opt=='A')
        printf("%c\n",buf[(pos+x-1)%len]);
        else
        pos=(pos+len+x)%len;
    }
    return 0;
}
```

## 8、Fraction Constructive Problem

链接

### 题意

若干询问，每次询问给定 $a,b(1\le a,b\le 2\times 10^6)$，要求构造 $1\le c,d,e,f\le 4\times 10^{12}$，

构造需要满足 $d,f\lt b$ 或 $\frac cd-\frac ef=\frac ab$。

### 题解

情况 $1$:$(a,b)\gt 1$

记 $a'=\frac a{(a,b)},b'=\frac b{(a,b)}$，令 $c=a'+1,d=b',e=1,f=b'$ 即可。

情况 $2$:$(a,b)== 1$ 且 $b\neq p^\alpha$

令 $b=df$ 且 $(d,f)=1$，根据裴蜀定理和拓展欧几里得算法，可以找到 $fx+dy=1$。

于是有 $\frac xd+\frac yf=\frac 1b$，令 $c=ax,e=ay$ 即可，需要注意调整负号。

关于 $b=df$ 中合适的 $d,f$ 的寻找，可以提前线性筛记录每个数最小素因子，令 $d$ 为 $b$ 最小素因子的幂次。

情况 $3$:$(a,b)== 1$ 且 $b= p^\alpha$(包括 $b=1$)

该情况无解，因为 $\frac cd-\frac ef$ 的分母为 $\text{lcm}(d,f)$ 的因子，而由于 $d,f\lt b$ 且 $b=p^\alpha$，必有 $\text{lcm}(d,f)$ 中 $p$ 因子幂次小于 $b$。

```
const int MAXP=2e6+5;
int vis[MAXP],prime[MAXP],cnt;
void Prime(){
    vis[1]=1;
    _for(i,2,MAXP){
        if(!vis[i])prime[cnt++]=i,vis[i]=i;
        for(int j=0;j<cnt&&i*prime[j]<MAXP;j++){
            vis[i*prime[j]]=prime[j];
            if(i%prime[j]==0) break;
        }
    }
}
LL gcd(LL a,LL b){
    while(b){
        LL t=b;
        b=a%b;
        a=t;
    }
    return  a;
}
LL exgcd(LL a,LL b,LL &tx,LL &ty){
    if(b==0){
        tx=1,ty=0;
        return a;
    }
    LL re=exgcd(b,a%b,ty,tx);
    ty-=a/b*tx;
    return re;
}
int main()
{
    int t=read_int();
    Prime();
    while(t--){
        int a=read_int(),b=read_int();LL c,d,e,f,temp;
        if((temp=gcd(a,b))>1){
```

```
            d=f=b/temp;
            e=1;
            c=a/temp+1;
        }
        else{
            d=1,f=b;
            while(f!=1&&f%vis[b]==0)
            d*=vis[b],f/=vis[b];
            if(f==1){
                puts("-1 -1 -1 -1");
                continue;
            }
            exgcd(f,d,c,e);
            if(c<0)
            swap(c,e),swap(d,f);
            e=-e;
            c*=a,e*=a;
        }
        printf("%lld %lld %lld %lld\n",c,d,e,f);
    }
    return 0;
}
```

# 9□Operation on a Graph

链接

## 题意

给定一个图，图中点 $i$ 初始颜色为 $i$□

每次选择一种颜色的点进行操作，每次操作将与该颜色相邻的点所属颜色的所有点染成该颜色。(如果所选颜色不存在则忽略操作)

问经过若干次操作后所有点最终的颜色。

## 题解

并查集维护颜色，链表维护与每种颜色节点相连的节点，每个节点最多向外合并一次，于是每个链表节点最多遍历一次，线性复杂度。

```
const int MAXN=8e5+5;
struct Edge{
    int to,next;
}edge[MAXN<<1];
int head[MAXN],tail[MAXN],edge_cnt;
void Insert(int u,int v){
```

```
    edge[++edge_cnt]=Edge{v,head[u]};
    if(!head[u])tail[u]=edge_cnt;
    head[u]=edge_cnt;
}
int p[MAXN];
int Find(int x){return x==p[x]?x:p[x]=Find(p[x]);}
int main()
{
    int t=read_int();
    while(t--){
        int n=read_int(),m=read_int(),u,v;
        _for(i,0,n)
        p[i]=i,head[i]=tail[i]=0;
        while(m--){
            u=read_int(),v=read_int();
            Insert(u,v);Insert(v,u);
        }
        int q=read_int();
        while(q--){
            u=read_int();
            if(Find(u)!=u)
            continue;
            int temp_head=0,temp_tail=0;
            for(int i=head[u];i;i=edge[i].next){
                v=Find(edge[i].to);
                if(v==u)continue;
                edge[tail[v]].next=temp_head;
                if(!temp_head)temp_tail=tail[v];
                temp_head=head[v];
                p[v]=u;
            }
            head[u]=temp_head;tail[u]=temp_tail;
        }
        _for(i,0,n)
        space(Find(i));
        puts("");
    }
    return 0;
}
```