

E. Contamination

题意

二维平面中给定 n 个圆。接下来 q 个询问，每次询问给定 $P(x,y), Q(x,y), y_1, y_2$ 问 P 是否可以移动到 Q

移动过程中不能进入圆的范围且 y 始终在 $[y_1, y_2]$ 保证 P, Q 一定不属于任意一个圆，且任意两圆都相离。

题解

不妨设 $P_x \leq Q_x$ 不难发现 P 可以移动到 Q 的充要条件为不存在一个圆 (x,y,r) 满足 $P_x \leq x \leq Q_x$ 且 $y-r \leq y_1, y+r \leq y_2$

考虑扫描线维护答案，将所有询问按 y 排序，对每个圆，在 $y=y_i-r_i$ 时加入贡献 $y=y_i+r_i$ 时删除贡献。

线段树维护 X 轴区间上每个位置的有效的圆的 $y+r$ 的最大值。于是题目转化为单点修改、区间查询。

对每个叶子额外开一个多重集维护该位置的 $y+r$ 的最大值即可，时间复杂度 $O((n+q)\log n)$

```

const int MAXN=1e6+5,MAXQ=1e6+5,inf=2e9+5;
int lef[MAXN<<2],rig[MAXN<<2],s[MAXN<<2];
multiset<int> num[MAXN<<2];
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R,s[k]=-inf;
    if(L==R)
        return;
    int M=L+R>>1;
    build(k<<1,L,M);
    build(k<<1|1,M+1,R);
}
void update(int k,int pos,int v,bool add){
    if(lef[k]==rig[k]){
        if(add){
            num[k].insert(v);
            s[k]=*num[k].rbegin();
        }
        else{
            num[k].erase(num[k].find(v));
            if(num[k].empty())
                s[k]=-inf;
            else
                s[k]=*num[k].rbegin();
        }
    }
    return;
}

```

```
int mid=lef[k]+rig[k]>>1;
if(mid>=pos)
update(k<<1,pos,v,add);
else
update(k<<1|1,pos,v,add);
s[k]=max(s[k<<1],s[k<<1|1]);
}
int query(int k,int L,int R){
if(L<=lef[k]&&rig[k]<=R)
return s[k];
int mid=lef[k]+rig[k]>>1;
if(mid>=R)
return query(k<<1,L,R);
else if(mid<L)
return query(k<<1|1,L,R);
else
return max(query(k<<1,L,R),query(k<<1|1,L,R));
}
struct Node{
int type,y,my;
int idx,v1,v2;
Node(int type=0,int y=0,int my=0,int idx=0,int xl=0,int xr=0){
this->type=type;
this->y=y;
this->my=my;
this->idx=idx;
v1=xl;
v2=xr;
}
bool operator < (const Node &b)const{
if(y!=b.y)
return y<b.y;
else
return type<b.type;
}
}node[MAXN*2+MAXQ];
bool ans[MAXQ];
vector<int> mp;
int main(){
int n=read_int(),q=read_int(),m=0;
_for(i,0,n){
int cx=read_int(),cy=read_int(),r=read_int();
node[m++]=Node(0,cy-r,cy+r,cx);
node[m++]=Node(2,cy+r,cy+r,cx);
mp.push_back(cx);
}
_for(i,0,q){
int
px=read_int(),py=read_int(),qx=read_int(),qy=read_int(),ymin=read_int(),yma
x=read_int();
node[m++]=Node(1,ymin,ymax,i,min(px,qx),max(px,qx));
```

```

}
sort(mp.begin(),mp.end());
mp.erase(unique(mp.begin(),mp.end()),mp.end());
build(1,1,mp.size());
sort(node,node+m);
_for(i,0,m){
    Node opt=node[i];
    if(opt.type==0||opt.type==2){
        int p=lower_bound(mp.begin(),mp.end(),opt.idx)-mp.begin()+1;
        update(1,p,opt.my,opt.type==0);
    }
    else{
        int p1=lower_bound(mp.begin(),mp.end(),opt.v1)-mp.begin()+1;
        int p2=upper_bound(mp.begin(),mp.end(),opt.v2)-mp.begin();
        if(p1>p2)
            ans[opt.idx]=true;
        else
            ans[opt.idx]=(query(1,p1,p2)<opt.my);
    }
}
_for(i,0,q){
    if(ans[i])
        puts("YES");
    else
        puts("NO");
}
return 0;
}
}

```

From: <https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link: https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:%E7%BB%84%E9%98%9F%E8%AE%AD%E7%BB%B3%E6%AF%94%E8%B5%9B%E8%AE%B0%E5%BD%95%E7%BC%93%E5%86%B2%E5%8C%BA&rev=1631415192

Last update: 2021/09/12 10:53