整体二分

算法简介

一种同时对所有询问进行二分答案的离线算法，时间复杂度一般为 $O(n\log n\log v)$。

算法例题

动态区间第 k 小

洛谷p2617

题意

给定一个长度为 n 的序列，支持两种操作：

1. 表示查询下标在区间 $[l,r]$ 中的第 k 小的数
2. 将位置 x 的数值修改为 y

题解

先考虑没有操作 2 的情况。

可以二分答案 mid，使用树状数组维护序列前 x 个位置中有多少个数不超过 mid，由此可以快速计算出每个询问区间中 mid 的名次 r。

对每个询问 i，如果 $k_i\le r$，则说明询问 i 答案在左区间，直接转移到左区间处理。

否则询问 i 答案在右区间，于是将 k_i 减去 r，转移到右区间处理。

显然不能二分过程中总是遍历整个序列，考虑转移询问的同时将序列的每个元素也转移到对应的区间。

具体的，如果元素 $v\le mid$，则转移到左区间，否则转移到右区间。

于是不考虑操作 2 的情况下的问题已经得到解决。

接下来考虑如何处理操作 2，发现每个操作 2 可以理解为删除和插入，而序列元素的初值可以考虑为直接插入。

于是考虑不是将每个序列元素转移，而是将每个修改操作转移，同时也用树状数组维护。

考虑到修改对查询会产生影响，于是需要注意维护时间序的相对不变性，具体见代码。

```
const int MAXN=1e5+5,Inf=1e9;
struct OPT{
    int type,x,y,k,id;
```

```
}opt[MAXN<<2],temp1[MAXN<<2],temp2[MAXN<<2];
int n,a[MAXN],c[MAXN],ans[MAXN];
#define lowbit(x) ((x)&(-x))
void add(int pos,int v){
    while(pos<=n){
        c[pos]+=v;
        pos+=lowbit(pos);
    }
}
int query(int pos){
    int ans=0;
    while(pos){
        ans+=c[pos];
        pos-=lowbit(pos);
    }
    return ans;
}
void solver(int ql,int qr,int vl,int vr){
    if(ql>qr)return;
    int vm=vl+vr>>1;
    if(vl==vr){
        _rep(i,ql,qr) if(opt[i].type==1)
        ans[opt[i].id]=vm;
        return;
    }
    int cnt1=0,cnt2=0,pos=ql;
    _rep(i,ql,qr){
        if(opt[i].type==0){
            if(opt[i].y<=vm){
                add(opt[i].x,opt[i].k);
                temp1[++cnt1]=opt[i];
            }
            else
            temp2[++cnt2]=opt[i];
        }
        else{
            int cnt=query(opt[i].y)-query(opt[i].x-1);
            if(opt[i].k<=cnt)
            temp1[++cnt1]=opt[i];
            else{
                temp2[++cnt2]=opt[i];
                temp2[cnt2].k-=cnt;
            }
        }
    }
    _rep(i,1,cnt1){
        if(temp1[i].type==0)add(temp1[i].x,-temp1[i].k);
        opt[pos++]=temp1[i];
    }
    _rep(i,1,cnt2)opt[pos++]=temp2[i];
```

```
    solver(ql,qr-cnt2,vl,vm);solver(ql+cnt1,qr,vm+1,vr);
}
int main()
{
    n=read_int();
    int m=read_int(),opt_cnt=0,q_cnt=0,x,y;
    _rep(i,1,n){
        a[i]=read_int();
        opt[++opt_cnt]=OPT{0,i,a[i],1,0};
    }
    _rep(i,1,m){
        char c=get_char();x=read_int(),y=read_int();
        if(c=='C'){
            opt[++opt_cnt]=OPT{0,x,a[x],-1,0};
            opt[++opt_cnt]=OPT{0,x,a[x]=y,1,0};
        }
        else
        opt[++opt_cnt]=OPT{1,x,y,read_int(),++q_cnt};
    }
    solver(1,opt_cnt,0,Inf);
    _rep(i,1,q_cnt)
    enter(ans[i]);
    return 0;
}
```

算法练习

习题一

洛谷p1527

题意

给定一个长度为 $n\times n$ 矩阵，q 次询问。每次询问子矩阵中第 k 大元素。

题解

参考算法例题中没有修改操作的一维序列区间第 k 大做法，考虑二分过程中转移元素和询问。

使用二维树状数组维护名次，总时间复杂度 $O\left(n^2\log^3 n\right)$。

```
const int MAXN=505,MAXQ=6e4+5;
struct Num{
    int x,y,v;
    bool operator < (const Num &b){
        return v<b.v;
```

```
    }
}a[MAXN*MAXN];
struct Query{
    int x1,y1,x2,y2,k,id;
}q[MAXQ],q1[MAXQ],q2[MAXQ];
int n,c[MAXN][MAXN],ans[MAXQ];
#define lowbit(x) ((x)&(-x))
void add(int x,int y,int v){
    for(int i=x;i<=n;i+=lowbit(i))
    for(int j=y;j<=n;j+=lowbit(j))
    c[i][j]+=v;
}
int get_s(int x,int y){
    int ans=0;
    for(int i=x;i;i-=lowbit(i))
    for(int j=y;j;j-=lowbit(j))
    ans+=c[i][j];
    return ans;
}
int query(int x1,int y1,int x2,int y2){
    return get_s(x2,y2)-get_s(x1-1,y2)-get_s(x2,y1-1)+get_s(x1-1,y1-1);
}
void solver(int ql,int qr,int vl,int vr){
    if(ql>qr)return;
    int vm=vl+vr>>1;
    if(vl==vr){
        _rep(i,ql,qr)ans[q[i].id]=a[vm].v;
        return;
    }
    _rep(i,vl,vm)add(a[i].x,a[i].y,1);
    int cnt1=0,cnt2=0,pos=ql;
    _rep(i,ql,qr){
        int r=query(q[i].x1,q[i].y1,q[i].x2,q[i].y2);
        if(q[i].k<=r)q1[++cnt1]=q[i];
        else{
            q2[++cnt2]=q[i];
            q2[cnt2].k-=r;
        }
    }
    _rep(i,vl,vm)add(a[i].x,a[i].y,-1);
    _rep(i,1,cnt1)q[pos++]=q1[i];
    _rep(i,1,cnt2)q[pos++]=q2[i];
    solver(ql,qr-cnt2,vl,vm);solver(ql+cnt1,qr,vm+1,vr);
}
int main()
{
    n=read_int();
    int m=read_int(),cnt=0,x1,y1,x2,y2;
    _rep(i,1,n)_rep(j,1,n)
    a[++cnt]=Num{i,j,read_int()};
```

```
    sort(a+1,a+cnt+1);
    _rep(i,1,m){
        x1=read_int(),y1=read_int(),x2=read_int(),y2=read_int();
        q[i]=Query{x1,y1,x2,y2,read_int(),i};
    }
    solver(1,m,1,cnt);
    _rep(i,1,m)
    enter(ans[i]);
    return 0;
}
```