

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 $k_i \leq r$ 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 $v \leq mid$ 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,vn);solver(ql+cnt1,qr,vn+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 $\square$ 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 $O(n^2 \log^3 n)$

```

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;
}

```

习题二

[洛谷p3527](#)

题意

给定一个长度为 m 的环，每个节点有一个所属国家 k 次事件，每次对 $[l,r]$ 顺时针环形区间上的每个点点权加上一个值。

每个国家有一个期望值 p_i 求每个国家最早多少次操作之后属于该国家的所有点的点权和能不小于 p_i

题解

首先把考虑如何把环上修改转化成线段修改，发现将每个 $[l,r](l < r)$ 事件拆成 $[l,m],[1,r]$ 事件即可。

接下来直接整体二分，计算贡献时差分 $++$ 树状数组维护区间修改 $++$ 单点查询操作，每个国家暴力查询属于他的点集的点权和。

发现每层修改时间复杂度 $O(m \log m)$ 查询复杂度也是 $O(m \log m)$ 共 $O(\log m)$ 层。

于是总时间复杂度 $O(m \log^2 m)$ 事实上本题存在 $O(m \log m)$ 做法，有兴趣的可以自行思考。

```

const int MAXN=3e5+5,Inf=1e9;
struct Num{
    int l,r,v,t;
}a[MAXN<<1];
vector<int> ch[MAXN];
struct Query{
    int k,id;
}q[MAXN],q1[MAXN],q2[MAXN];
int m,ans[MAXN];
LL c[MAXN];
#define lowbit(x) ((x)&(-x))
void add(int lef,int rig,int v){
    int pos=lef;

```

```
while(pos<=m){
    c[pos]+=v;
    pos+=lowbit(pos);
}
pos=rig+1;
while(pos<=m){
    c[pos]-=v;
    pos+=lowbit(pos);
}
}
LL query(int idx){
    LL ans=0;
    _for(i,0,ch[idx].size()){
        int pos=ch[idx][i];
        while(pos){
            ans+=c[pos];
            pos-=lowbit(pos);
        }
        if(ans>Inf)
            return ans;
    }
    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) ans[q[i].id]=a[vm].t;
        return;
    }
    _rep(i,vl,vm) add(a[i].l,a[i].r,a[i].v);
    int cnt1=0,cnt2=0,pos=ql;
    _rep(i,ql,qr){
        LL r=query(q[i].id);
        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].l,a[i].r,-a[i].v);
    _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()
{
    int n=read_int(),cnt=0,lef,rig,v;
    m=read_int();
    _rep(i,1,m)
```

```
ch[read_int()].push_back(i);
_rep(i,1,n)
q[i]=Query{read_int(),i};
int k=read_int();
_rep(i,1,k){
    lef=read_int(),rig=read_int(),v=read_int();
    if(lef<=rig)
        a[++cnt]=Num{lef,rig,v,i};
    else{
        a[++cnt]=Num{lef,m,v,i};
        a[++cnt]=Num{1,rig,v,i};
    }
}
solver(1,n,1,cnt+1);
_rep(i,1,n){
    if(!ans[i])
        puts("NIE");
    else
        enter(ans[i]);
}
return 0;
}
```

From:

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

Permanent link:

https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:%E6%95%B4%E4%BD%93%E4%BA%8C%E5%88%86

Last update: 2020/07/30 18:48