CCPC Wannafly Camp Day1

比赛链接

I. K 小数查询

题意

给定一个长度为 n 的序列，接下来 q 个操作：

1. 选取区间 $[l,r]$$a_i=\min(a_i,v)(l\le i\le r)$
2. 询问区间 $[l,r]$ 的第 k 小元素

题解

建立线段树套权值线段树。对于查询操作，考虑二分答案，然后查询区间中小于二分值的数个数判定答案合法性，时间复杂度 $O(\log^3 n)$

对于修改操作，考虑在线段树中找到对应区间，然后暴力修改该区间及其所有祖先结点对应的权值线段树。

修改完成后同样打上懒标记，必要时下放标记。

显然每次修改复杂度为 $O(k\log n)$其中 k 为该次修改删除的权值线段树的叶子数。

由于初始时仅有 $O(n\log n)$ 个权值线段树的叶子结点，于是修改操作的总时间复杂度为 $O(n\log^2 n)$

于是总时间复杂度 $O(n\log^3 n)$总空间复杂度 $O(n\log^2 n)$

```
const int MAXN=8e4+5,MAXM=400,Inf=1e9;
int a[MAXN],root[MAXN<<2],lef[MAXN<<2],rig[MAXN<<2],lazy[MAXN<<2];
int n,sz,s[MAXN*MAXM],ch[MAXN*MAXM][2];
void update_1D(int &k,int pos,int v,int lef=1,int rig=n){
    if(!k)k=++sz;
    s[k]+=v;
    if(lef==rig)return;
    int mid=lef+rig>>1;
    if(pos<=mid)
    update_1D(ch[k][0],pos,v,lef,mid);
    else
    update_1D(ch[k][1],pos,v,mid+1,rig);
}
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R,lazy[k]=Inf;
    _rep(i,L,R)update_1D(root[k],a[i],1);
    if(L==R)return;
```

```
    int M=L+R>>1;
    build(k<<1,L,M);
    build(k<<1|1,M+1,R);
}
vector<pair<int,int> >del;
void dfs(int k,int pos,int lef=1,int rig=n){
    if(!s[k])return;
    if(lef==rig){
        del.push_back(make_pair(lef,s[k]));
        return;
    }
    int mid=lef+rig>>1;
    if(pos<mid)dfs(ch[k][0],pos,lef,mid);
    dfs(ch[k][1],pos,mid+1,rig);
}
void dfs2(int k,int pos,int lef=1,int rig=n){
    if(!s[k])return;
    if(lef==rig){
        s[k]=0;
        return;
    }
    int mid=lef+rig>>1;
    if(pos<mid)dfs2(ch[k][0],pos,lef,mid);
    dfs2(ch[k][1],pos,mid+1,rig);
    s[k]=s[ch[k][0]]+s[ch[k][1]];
}
void update_v(int k,int v){
    if(s[root[k]]!=rig[k]-lef[k]+1)
    update_1D(root[k],v,rig[k]-lef[k]+1-s[root[k]]);
}
void push_down(int k){
    if(lazy[k]!=Inf){
        dfs2(root[k<<1],lazy[k]);
        update_v(k<<1,lazy[k]);
        lazy[k<<1]=min(lazy[k<<1],lazy[k]);
        dfs2(root[k<<1|1],lazy[k]);
        update_v(k<<1|1,lazy[k]);
        lazy[k<<1|1]=min(lazy[k<<1|1],lazy[k]);
        lazy[k]=Inf;
    }
}
void update_2D(int k,int L,int R,int v){
    if(L<=lef[k]&&rig[k]<=R){
        lazy[k]=min(lazy[k],v);
        del.clear();
        dfs(root[k],v);
        while(k){
            _for(i,0,del.size())
            update_1D(root[k],del[i].first,-del[i].second);
            update_v(k,v);
```

```
            k>>=1;
        }
        return;
    }
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    if(mid>=L)update_2D(k<<1,L,R,v);
    if(mid<R)update_2D(k<<1|1,L,R,v);
}
int query_1D(int k,int L,int R,int lef=1,int rig=n){
    if(!k)return 0;
    if(L<=lef&&rig<=R)return s[k];
    int mid=lef+rig>>1;
    if(mid>=R)return query_1D(ch[k][0],L,R,lef,mid);
    else if(mid<L)return query_1D(ch[k][1],L,R,mid+1,rig);
    return query_1D(ch[k][0],L,R,lef,mid)+query_1D(ch[k][1],L,R,mid+1,rig);
}
int query_2D(int k,int L,int R,int v){
    if(L<=lef[k]&&rig[k]<=R)
    return query_1D(root[k],1,v);
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    if(mid>=R)return query_2D(k<<1,L,R,v);
    else if(mid<L)return query_2D(k<<1|1,L,R,v);
    return query_2D(k<<1,L,R,v)+query_2D(k<<1|1,L,R,v);
}
int main()
{
    n=read_int();
    int m=read_int();
    _rep(i,1,n)a[i]=read_int();
    build(1,1,n);
    while(m--){
        int opt=read_int(),l=read_int(),r=read_int(),v=read_int();
        if(opt==1){
            if(v>=n)continue;
            update_2D(1,l,r,v);
        }
        else{
            int lef=1,rig=n,mid,ans=-1;
            while(lef<=rig){
                mid=lef+rig>>1;
                if(query_2D(1,l,r,mid)>=v){
                    ans=mid;
                    rig=mid-1;
                }
                else
                lef=mid+1;
            }
            enter(ans);
        }
```

```
    }
    return 0;
}
```