数据结构练习 1

线段树合并/分裂

习题一

洛谷p2824

题意

给定由 $1\sim n$ 的排列构成的序列 A，要求支持以下下两种操作：

1. 将 $A_{l\sim r}$ 按升序排列
2. 将 $A_{l\sim r}$ 按降序排列

m 次操作后询问位置 p 上的数值。

题意

考虑初始时构造 n 棵权值线段树，每个线段树维护区间 $[i,i](1\le i\le n)$。

每次排序操作将所有代表区间含于 $[l,r]$ 的线段树合并，发现合并完成的同时也完成了排序。

但由于区间 $[l,b]$ 可能已经属于某个区间 $[a,b]$，所以需要线段树分裂将其分裂为 $[a,l-1],[l,b]$。

同样也需要将区间 $[c,d]$ 分裂为 $[c,r],[r+1,d]$，最后将 $[l,b]\cdots [c,r]$ 等区间合并即可，注意打标记维护区间的升序/降序情况。

初始化时间复杂度 $O(n\log n)$，产生点数 $O(n\log n)$，每次分裂时间复杂度 $O(\log n)$，且增加 $O(\log n)$ 个点。

每次合并操作时间复杂度等于合并的点数，于是一定不会超过初始化和分裂生成的总点数。

对于查询操作，考虑单独将区间 $[p,p]$ 分裂出来然后 dfs 沿唯一路径到达叶子结点即可，时间复杂度 $O(\log n)$。

考虑 ODT 维护区间，总时空间复杂度 $O((n+m)\log n)$。

```
const int MAXN=1e5+5,MAXM=60;
struct Node{
    int ch[2],cnt;
}node[MAXN*MAXM];
int root[MAXN],node_cnt;
void push_up(int
k){node[k].cnt=node[node[k].ch[0]].cnt+node[node[k].ch[1]].cnt;}
void update(int &k,int lef,int rig,int pos){
```

```
    k=++node_cnt;
    node[k].cnt++;
    if(lef==rig)return;
    int mid=lef+rig>>1;
    if(mid>=pos)
    update(node[k].ch[0],lef,mid,pos);
    else
    update(node[k].ch[1],mid+1,rig,pos);
}
int query(int k,int lef,int rig){
    int mid=lef+rig>>1;
    if(lef==rig)return mid;
    if(node[k].ch[0])
    return query(node[k].ch[0],lef,mid);
    else
    return query(node[k].ch[1],mid+1,rig);
}
void Merge(int &k1,int k2,int lef,int rig){
    if(!k1||!k2) return k1|=k2,void();
    if(lef==rig) return node[k1].cnt+=node[k2].cnt,void();
    int mid=lef+rig>>1;
    Merge(node[k1].ch[0],node[k2].ch[0],lef,mid);
    Merge(node[k1].ch[1],node[k2].ch[1],mid+1,rig);
    push_up(k1);
}
void split(int &k1,int &k2,int k){
    if(node[k1].cnt==k)return;
    k2=++node_cnt;
    if(k<=node[node[k1].ch[0]].cnt){
        node[k2].ch[1]=node[k1].ch[1];
        node[k1].ch[1]=0;
        split(node[k1].ch[0],node[k2].ch[0],k);
    }
    else
    split(node[k1].ch[1],node[k2].ch[1],k-node[node[k1].ch[0]].cnt);
    push_up(k1);push_up(k2);
}
struct seg{
    int lef,rig,rev;
    bool operator < (const seg &b)const{
        return lef<b.lef;
    }
    seg(int lef,int rig,int rev):lef(lef),rig(rig),rev(rev){}
    seg(int pos){lef=pos;}
};
set<seg> s;
typedef set<seg>::iterator iter;
iter split2(int pos){
    iter it=s.lower_bound(seg(pos));
    if(it!=s.end()&&it->lef==pos)return it;
```

```
    --it;
    int lef=it->lef,rig=it->rig,rev=it->rev;
    if(rev){
        split(root[lef],root[pos],rig-pos+1);
        swap(root[lef],root[pos]);
    }
    else
    split(root[lef],root[pos],pos-lef);
    s.erase(it);s.insert(seg(lef,pos-1,rev));
    return s.insert(seg(pos,rig,rev)).first;
}
int main()
{
    int n=read_int(),m=read_int();
    _rep(i,1,n)update(root[i],1,n,read_int()),s.insert(seg(i,i,0));
    while(m--){
        int opt=read_int(),lef=read_int(),rig=read_int();
        iter r=split2(rig+1),l=split2(lef);
        for(iter it=++l;it!=r;++it)Merge(root[lef],root[it->lef],1,n);
        s.erase(--l,r);
        s.insert(seg(lef,rig,opt));
    }
    int p=read_int();
    split2(p+1);split2(p);
    enter(query(root[p],1,n));
        return 0;
}
```