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;
}
```

习题二

洛谷p5298

题意

给定一棵以 1 为根的有根树，每个点有一个权值，且度数不超过 2。

若 i 为叶子结点，则 i 的权值等于 v□否则它的权值有 p 的概率为儿子结点中的较大值□$1-p$ 的概率为儿子节点中的较小值。

假设 1 号节点有 m 中可能取值□V_i 为所有可能值中第 i 小的取值□D_i 为取到第 i 小的取值的概率，询问

$$\sum_{i=1}^m iV_iD_i^2$$

数据保证所有叶子节点权值互异。

题解

先对所有权值进行离散化，设 $D(i,j)$ 表示节点 i 取值为 j 的概率。

对树进行 dfs，如果该节点是叶子节点，直接可知该结点 D 数组在该点权值处为 1，其余处为 0。

如果该节点只有一个子节点，则该节点可以直接继承子节点的 D 数组。

如果该节点有两个子节点，对取值 i，由于叶子节点取值互异，不难得到状态转移：

$$D(u,i)=\left(D(l,i)\left(p_i\sum_{j=1}^{i-1}D(r,j)+(1-p_i)\sum_{j=i+1}^mD(r,j)\right)+D(r,i)\left(p_i\sum_{j=1}^{i-1}D(l,j)+(1-p_i)\sum_{j=i+1}^mD(l,j)\right)\right)$$

考虑用权值线段树维护每个节点的 D 数组。对于叶子节点和只有一个子节点的节点，不难维护。

对于有两个子节点的节点和某个确定的取值 i，显然至多只有一个子节点满足该节点取值为 i 的概率不为 0。

于是不可能出现被合并的两棵线段树拥有重复的叶子节点的情况。于是合并的终止条件一定某个线段树该节点为空。

不妨设左节点的线段树非空，且当前线段树节点代表区间为 $[x,y]$，于是对 $i\in [x,y]$ 有

$$D(u,i)=D(l,i)\left(p_i\sum_{j=1}^{x-1}D(r,j)+(1-p_i)\sum_{j=y+1}^mD(r,j)\right)$$

考虑线段树合并过程中维护两棵线段树的区间前缀和 $[1,x-1]$ 和后缀和 $[y+1,m]$ 即可。

最后对 1 节点的线段树 dfs 即可得到答案。时空间复杂度 $O(n\log n)$。

```
const int MAXN=3e5+5,MAXM=60,Mod=998244353;
int quick_pow(int a,int b){
    int ans=1;
    while(b){
        if(b&1)ans=1LL*ans*a%Mod;
        a=1LL*a*a%Mod;
        b>>=1;
    }
    return ans;
}
const int inv=quick_pow(10000,Mod-2);
struct Node{
    int ch[2],s,lazy;
    void add_lazy(int v){
        s=1LL*s*v%Mod;
        lazy=1LL*lazy*v%Mod;
    }
}node[MAXN*MAXM];
int root[MAXN],node_cnt;
void push_down(int k){
    if(node[k].lazy!=1){
        node[node[k].ch[0]].add_lazy(node[k].lazy);
        node[node[k].ch[1]].add_lazy(node[k].lazy);
```

```
        node[k].lazy=1;
    }
}
void push_up(int
k){node[k].s=(node[node[k].ch[0]].s+node[node[k].ch[1]].s)%Mod;}
void update(int &k,int lef,int rig,int pos){
    k=++node_cnt;
    node[k].lazy=node[k].s=1;
    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 n,temp_p,p[MAXN],m;
void Merge(int &k1,int k2,int lpre,int lsuf,int rpre,int rsuf){
    if(!k1&&!k2)return;
    else if(k1&&k2){
        push_down(k1);push_down(k2);
        int t1=node[node[k1].ch[0]].s,t2=node[node[k2].ch[0]].s;
Merge(node[k1].ch[0],node[k2].ch[0],lpre,(lsuf+node[node[k1].ch[1]].s)%Mod,
rpre,(rsuf+node[node[k2].ch[1]].s)%Mod);
Merge(node[k1].ch[1],node[k2].ch[1],(lpre+t1)%Mod,lsuf,(rpre+t2)%Mod,rsuf);
        push_up(k1);
    }
    else{
        if(k1)node[k1].add_lazy((1LL*rpre*temp_p+1LL*rsuf*(Mod+1-
temp_p))%Mod);
        else node[k1=k2].add_lazy((1LL*lpre*temp_p+1LL*lsuf*(Mod+1-
temp_p))%Mod);
    }
}
struct Node_2{
    int ch[2],v;
}node2[MAXN];
void dfs(int k){
    if(!node2[k].ch[0])
    update(root[k],1,m,node2[k].v);
    else if(!node2[k].ch[1]){
        dfs(node2[k].ch[0]);
        root[k]=root[node2[k].ch[0]];
    }
    else{
        dfs(node2[k].ch[0]);dfs(node2[k].ch[1]);
        temp_p=node2[k].v;
        root[k]=root[node2[k].ch[0]];
        Merge(root[k],root[node2[k].ch[1]],0,0,0,0);
    }
}
int dfs2(int k,int lef,int rig){
```

```
    if(!k)return 0;
    push_down(k);
    int mid=lef+rig>>1;
    if(lef==rig)return 1LL*mid*p[mid]%Mod*node[k].s%Mod*node[k].s%Mod;
    return (dfs2(node[k].ch[0],lef,mid)+dfs2(node[k].ch[1],mid+1,rig))%Mod;
}
int main()
{
    n=read_int();
    _rep(i,1,n){
        int f=read_int();
        if(f){
            if(node2[f].ch[0])node2[f].ch[1]=i;
            else node2[f].ch[0]=i;
        }
    }
    _rep(i,1,n){
        node2[i].v=read_int();
        if(node2[i].ch[0])node2[i].v=1LL*node2[i].v*inv%Mod;
        else p[++m]=node2[i].v;
    }
    sort(p+1,p+m+1);
    m=unique(p+1,p+m+1)-p;
    _rep(i,1,n)if(!node2[i].ch[0])
    node2[i].v=lower_bound(p+1,p+m,node2[i].v)-p;
    dfs(1);
    enter(dfs2(1,1,m));
    return 0;
}
```