吉司机线段树

算法简介

一种支持区间最值修改及各种查询的线段树。

算法模板

模板一

HDU5306

题意

给定一个序列，支持下述三种操作：

1. 对 $l\le i\le r$，$a_i=\min(a_i,v)$
2. 询问区间 $[l,r]$ 的最大值
3. 询问区间 $[l,r]$ 的和

题解

考虑建立线段树，维护区间最大值，区间最大值个数，区间次大值，区间和。

对修改操作，如果v不小于区间最大值，则该操作没有意义，直接返回，否则先进行普通区间修改操作。

当普通区间修改操作遇到当前区间被包含于修改区间时，如果 v 的大于区间次大值，则直接修改区间最大值，同时打上懒标记。

如果 v 不大于区间次大值，则暴力修改子树。发现修改过程本质是 dfs，而 dfs 时间复杂度等于遍历结点个数。

定义一个结点所代表区间的不同数值所构成的集合为该结点的数集，则暴力修改至少会合并该结点的最大值和次大值，于是该结点数集大小减小。

普通区间修改操作至多会使该结点的数集增加一个值，但普通区间修改的结点只有 $O(\log n)$ 个。

初始时所有结点的数集大小和为 $O(n\log n)$，于是修改操作的总复杂度为 $O((n+m)\log n)$。

对查询操作，和普通区间查询操作完全相同。

```
const int MAXN=1e6+5;
int a[MAXN];
int
lef[MAXN<<2],rig[MAXN<<2],maxv[MAXN<<2],maxc[MAXN<<2],secv[MAXN<<2],lazy[MA
XN<<2];
```

```
LL sum[MAXN<<2];
void push_up(int k){
    sum[k]=sum[k<<1]+sum[k<<1|1];
    if(maxv[k<<1]==maxv[k<<1|1]){
        maxv[k]=maxv[k<<1];
        maxc[k]=maxc[k<<1]+maxc[k<<1|1];
        secv[k]=max(secv[k<<1],secv[k<<1|1]);
    }
    else if(maxv[k<<1]>maxv[k<<1|1]){
        maxv[k]=maxv[k<<1];
        maxc[k]=maxc[k<<1];
        secv[k]=max(secv[k<<1],maxv[k<<1|1]);
    }
    else{
        maxv[k]=maxv[k<<1|1];
        maxc[k]=maxc[k<<1|1];
        secv[k]=max(maxv[k<<1],secv[k<<1|1]);
    }
}
void push_tag(int k,int v){
    if(v>=maxv[k])return;
    sum[k]-=1LL*maxc[k]*(maxv[k]-v);
    maxv[k]=lazy[k]=v;
}
void push_down(int k){
    if(~lazy[k]){
        push_tag(k<<1,lazy[k]);
        push_tag(k<<1|1,lazy[k]);
        lazy[k]=-1;
    }
}
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R,lazy[k]=-1;
    int M=L+R>>1;
    if(L==R){
        sum[k]=maxv[k]=a[M];
        secv[k]=-1;
        maxc[k]=1;
        return;
    }
    build(k<<1,L,M);
    build(k<<1|1,M+1,R);
    push_up(k);
}
void update(int k,int L,int R,int v){
    if(maxv[k]<=v)return;
    if(L<=lef[k]&&rig[k]<=R&&secv[k]<v)
    return push_tag(k,v);
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
```

```
    if(mid>=L)update(k<<1,L,R,v);
    if(mid<R)update(k<<1|1,L,R,v);
    push_up(k);
}
int query_max(int k,int L,int R){
    if(L<=lef[k]&&rig[k]<=R)return maxv[k];
    push_down(k);
    int mid=lef[k]+rig[k]>>1,vl=-1,vr=-1;
    if(mid>=L)vl=query_max(k<<1,L,R);
    if(mid<R)vr=query_max(k<<1|1,L,R);
    return max(vl,vr);
}
LL query_sum(int k,int L,int R){
    if(L<=lef[k]&&rig[k]<=R)return sum[k];
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    LL s=0;
    if(mid>=L)s+=query_sum(k<<1,L,R);
    if(mid<R)s+=query_sum(k<<1|1,L,R);
    return s;
}
int main()
{
    int T=read_int();
    while(T--){
        int n=read_int(),q=read_int();
        _rep(i,1,n)a[i]=read_int();
        build(1,1,n);
        int opt,x,y;
        while(q--){
            opt=read_int(),x=read_int(),y=read_int();
            if(opt==0)
            update(1,x,y,read_int());
            else if(opt==1)
            enter(query_max(1,x,y));
            else
            enter(query_sum(1,x,y));
        }
    }
    return 0;
}
```