19. 最值线段树

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll INF = 0x7fffffffffffffff;

// 最值线段树 实现log^2n的区间增加、区间最值、区间查询
class MxSegTree{
    struct data{
        ll mx,mx2,cmx,tmx;
        ll mn,mn2,cmn,tmn;
        ll tad,sum;
        data(){mx=mx2=-INF;mn=mn2=INF;cmx=cmn=0;tmx=-INF;tmn=INF;tad=0;sum=0;}
    };
    vector<data> tree;
    vector<ll> *arr;
    long n,n5,root,end;

    void push_add(ll cl, ll cr, ll p,ll v){
        tree[p].sum += (cr - cl + 1) * v;
        tree[p].mx += v, tree[p].mn += v;
        if (tree[p].mx2 != -INF) tree[p].mx2 += v;
        if (tree[p].mn2 != INF) tree[p].mn2 += v;
        if (tree[p].tmx != -INF) tree[p].tmx += v;
        if (tree[p].tmn != INF) tree[p].tmn += v;
        tree[p].tad += v;
    }
    void push_min(ll p, ll tg){
        if (tree[p].mx <= tg) return;
        tree[p].sum += (tg * 1ll - tree[p].mx) * tree[p].cmx;
        if (tree[p].mn2 == tree[p].mx) tree[p].mn2 = tg;
        if (tree[p].mn == tree[p].mx) tree[p].mn = tg;
        if (tree[p].tmx > tg) tree[p].tmx = tg;
        tree[p].mx = tg, tree[p].tmn = tg;
    }
    void push_max(ll p, ll tg){
        if (tree[p].mn > tg) return;
        tree[p].sum += (tg * 1ll - tree[p].mn) * tree[p].cmn;
        if (tree[p].mx2 == tree[p].mn) tree[p].mx2 = tg;
        if (tree[p].mx == tree[p].mn) tree[p].mx = tg;
        if (tree[p].tmn < tg) tree[p].tmn = tg;
        tree[p].mn = tg, tree[p].tmx = tg;
    }
    void push_up(ll p) { //用子节点维护当前节点信息
        tree[p].sum = tree[p * 2].sum + tree[p * 2 + 1].sum;
        if (tree[p * 2].mx == tree[p * 2 + 1].mx) {
            tree[p].mx = tree[p * 2].mx, tree[p].cmx = tree[p * 2].cmx + tree[p *
2 + 1].cmx;
            tree[p].mx2 = max(tree[p * 2].mx2, tree[p * 2 + 1].mx2);
        } else if (tree[p * 2].mx > tree[p * 2 + 1].mx) {
            tree[p].mx = tree[p * 2].mx, tree[p].cmx = tree[p * 2].cmx;
            tree[p].mx2 = max(tree[p * 2].mx2, tree[p * 2 + 1].mx);
        } else {
            tree[p].mx = tree[p * 2 + 1].mx, tree[p].cmx = tree[p * 2 + 1].cmx;
            tree[p].mx2 = max(tree[p * 2].mx, tree[p * 2 + 1].mx2);
        }
        if (tree[p * 2].mn == tree[p * 2 + 1].mn) {
            tree[p].mn = tree[p * 2].mn, tree[p].cmn = tree[p * 2].cmn + tree[p *
2 + 1].cmn;
            tree[p].mn2 = min(tree[p * 2].mn2, tree[p * 2 + 1].mn2);
        } else if (tree[p * 2].mn < tree[p * 2 + 1].mn) {
            tree[p].mn = tree[p * 2].mn, tree[p].cmn = tree[p * 2].cmn;
            tree[p].mn2 = min(tree[p * 2].mn2, tree[p * 2 + 1].mn);
        } else {
            tree[p].mn = tree[p * 2 + 1].mn, tree[p].cmn = tree[p * 2 + 1].cmn;
            tree[p].mn2 = min(tree[p * 2].mn, tree[p * 2 + 1].mn2);
        }
    }
    void push_down(ll cl, ll cr, ll p) { //下放标记
        ll cm = cl + (cr - cl) / 2;
        if(tree[p].tad) push_add(cl, cm, p * 2, tree[p].tad), push_add(cm + 1, cr,
p * 2 + 1, tree[p].tad);
        if(tree[p].tmx != -INF) push_max(p * 2, tree[p].tmx), push_max(p * 2 + 1,
tree[p].tmx);
        if(tree[p].tmn != INF) push_min(p * 2, tree[p].tmn), push_min(p * 2 + 1,
tree[p].tmn);
        tree[p].tad = 0, tree[p].tmx = -INF, tree[p].tmn = INF;
    }

    void build(ll s, ll t, ll p) {
        tree[p].tmx = -INF, tree[p].tmn = INF;
        if (s == t) {
            tree[p].sum = tree[p].mx = tree[p].mn = (*arr)[s];
            tree[p].mx2 = -INF, tree[p].mn2 = INF;
            tree[p].cmx = tree[p].cmn = 1;
            return;
        }
        ll m = s + (t - s) / 2;
        build(s, m, p * 2);
        build(m + 1, t, p * 2 + 1);
        push_up(p);
    }

    void range_add(ll l, ll r, ll v, ll cl, ll cr, ll p) {
        if(cl > r || cr < l) return;
        if (l <= cl && cr <= r) return push_add(cl, cr, p, v);
        ll cm = cl + (cr - cl) / 2;
        push_down(cl, cr, p);
        if (l <= cm) range_add(l, r, v, cl, cm, p * 2);
        if (r > cm) range_add(l, r, v, cm + 1, cr, p * 2 + 1);
        push_up(p);
    }
    void range_min(ll l, ll r, ll v, ll cl, ll cr, ll p) {
        if(cl > r || cr < l || tree[p].mx <= v) return;
        if (l <= cl && cr <= r && tree[p].mx2 < v) return push_min(p, v);
        ll cm = cl + (cr - cl) / 2;
        push_down(cl, cr, p);
        if (l <= cm) range_min(l, r, v, cl, cm, p * 2);
        if (r > cm) range_min(l, r, v, cm + 1, cr, p * 2 + 1);
        push_up(p);
    }
    void range_max(ll l, ll r, ll v, ll cl, ll cr, ll p) {
        if(cl > r || cr < l || tree[p].mn >= v) return;
        if (l <= cl && cr <= r && tree[p].mn2 > v) return push_max(p, v);
        ll cm = cl + (cr - cl) / 2;
        push_down(cl, cr, p);
        if (l <= cm) range_max(l, r, v, cl, cm, p * 2);
        if (r > cm) range_max(l, r, v, cm + 1, cr, p * 2 + 1);
        push_up(p);
    }

    ll query_sum(ll l, ll r, ll cl, ll cr, ll p) {
        if(cl > r || cr < l) return 0;
        if (l <= cl && cr <= r) return tree[p].sum;
        ll cm = cl + (cr - cl) / 2;
        ll sum = 0;
        push_down(cl, cr, p);
        if (l <= cm) sum += query_sum(l, r, cl, cm, p * 2);
        if (r > cm) sum += query_sum(l, r, cm + 1, cr, p * 2 + 1);
        return sum;
    }
    ll query_max(ll l, ll r, ll cl, ll cr, ll p) {
        if(cl > r || cr < l) return -INF;
        if (l <= cl && cr <= r) return tree[p].mx;
        ll cm = cl + (cr - cl) / 2;
        ll mx = -INF;
        push_down(cl, cr, p);
        if (l <= cm) mx = max(mx, query_max(l, r, cl, cm, p * 2));
        if (r > cm) mx = max(mx, query_max(l, r, cm + 1, cr, p * 2 + 1));
        return mx;
    }
    ll query_min(ll l, ll r, ll cl, ll cr, ll p) {
        if(cl > r || cr < l) return INF;
        if (l <= cl && cr <= r) return tree[p].mn;
        ll cm = cl + (cr - cl) / 2;
        ll mn = INF;
        push_down(cl, cr, p);
        if (l <= cm) mn = min(mn, query_min(l, r, cl, cm, p * 2));
        if (r > cm) mn = min(mn, query_min(l, r, cm + 1, cr, p * 2 + 1));
        return mn;
    }

public:
    explicit MxSegTree(vector<ll> v) {
        n = v.size();
        n5 = n * 5;
        end = n - 1;
        root = 1;
        arr = &v;
        tree.resize(n5);
        build(0, end, 1);
        arr = nullptr;
    }
    void range_add(ll l, ll r, ll v) { range_add(l, r, v, 0, end, root); }
    void range_min(ll l, ll r, ll v) { range_min(l, r, v, 0, end, root); }
    void range_max(ll l, ll r, ll v) { range_max(l, r, v, 0, end, root); }
    ll query_sum(ll l, ll r) { return query_sum(l, r, 0, end, root); }
    ll query_max(ll l, ll r) { return query_max(l, r, 0, end, root); }
    ll query_min(ll l, ll r) { return query_min(l, r, 0, end, root); }
};
```

20. 数论分块

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;

vector<ll> vr; //数论分块，vr存储\floor{x/i}相同的分块的右端点
void Sqrt_dec(ll x){
    vr.clear(); ll r=1;
    for(ll l=1;l<=x;l=vr.back()+1){
        r=x/(x/l);
        vr.emplace_back(r);
    }
}

vector<ll> pref; //函数f前缀和
ll Get_G(ll x){ //数论分块，求函数g(x)=\sum_{i=1}^{x} f(i)*\floor{x/i}
    ll res=0,r=1;
    for(ll l=1;l<=x;l=vr.back()+1){
        r=x/(x/l);
        res+=(pref[r]-pref[l-1])*(x/l)*(r-l+1);
    }
    return res;
}
```