数据结构

ST 表

一维

```
vector<int> a(n+1);
// init a;
vector<vector<int>> st(20, vector<int>(n+1));
for(int i=1;i<=n;i++) st[0][i] = a[i];
for(int i=1;(1<<i)<=n;i++)
    for(int j=1;j+(1<<i)-1<=n;j++)
        st[i][j] = max(st[i-1][j], st[i-1][j+(1<<(i-1))]);
auto query = [&](int l,int r)
{
    int k = log2(r-l+1);
    return max(st[k][l], st[k][r-(1<<k)+1]);
};
```

二维

```
vector<vector<int>> a(n+1, vector<int>(m+1));
// init a
vector<vector<vector<int>>> st(10, vector<vector<int>>(n+1, vector<int>(m+1)));
for(int i=1;i<=n;i++)
    for(int j=1;j<=m;j++)
        st[0][i][j] = a[i][j];
for(int k=1;k<=20;k++)
    for(int i=1;i+(1<<k)-1<=n;i++)
        for(int j=1;j+(1<<k)-1<=m;j++)
            st[k][i][j] = min({
                st[k-1][i][j],
                st[k-1][i+(1<<(k-1))][j],
                st[k-1][i][j+(1<<(k-1))],
                st[k-1][i+(1<<(k-1))][j+(1<<(k-1))]
            });
auto query = [&](int r,int c,int s)  // (r, c) 为左上角的 s*s 方阵
{
    int k = log2(s);
    return min({
        st[k][r][c],
        st[k][r+s-(1<<k)][c],
        st[k][r][c+s-(1<<k)],
        st[k][r+s-(1<<k)][c+s-(1<<k)]
    });
};
```

单调队列

```
int a[maxn];
vector<int> solve(int n, int k, auto rule=less<int>())   //? less: min, greater: max
{
    vector<int> res; deque<int> q;
    for(int i=1;i<=n;i++)
    {
        if(!q.empty()&&q.front()+k==i) q.pop_front();
        while(!q.empty()&&rule(a[i],a[q.back()])) q.pop_back();
        q.push_back(i);
        if(i>=k) res.push_back(a[q.front()]);
    }
    return res;
}
```

单调栈

```
int a[maxn];
vector<int> solve(int n)
{
    vector<int> res; stack<int> st;
    for(int i=n;i>=1;i--) //? 反向遍历代表第 i 个元素之"后"
    {
        while(!st.empty()&&a[st.top()]<=a[i]) st.pop(); //? 小于等于号代表第一个"大于" a_i 的元素的下标
        res.push_back(st.empty()?0:st.top());
        st.push(i);
    }
    reverse(res.begin(), res.end());
    return res;
}
```

树状数组

```
int n; ll c[maxn];
int lowbit(int x) { return x&-x; }
void modify(int p,int x) { for(int i=p;i<=n;i+=lowbit(i)) c[i] += x; }
int query(int p)    // [1,p] 之和
{
    ll sum = 0;
    for(int i=p;i;i-=lowbit(i)) sum += c[i];
    return sum;
}
```

线段树分裂合并

```
struct Node { int l,r; ll val; } sgt[maxn*40];      //? 40 = 2*maxm*log2(maxn)
int cnt, root[maxn];
inline void pushup(int k) { sgt[k].val = sgt[sgt[k].l].val + sgt[sgt[k].r].val; }
void modify(int l,int r,int &k,int p,int x)     // 单点修改: p 位置的值加上 x, 空间复杂度 O(logn)
{
    if(!k) k = ++cnt;      // 如果到了 NULL 结点就新建一个
    sgt[k].val += x;     // 单点修改的加法直接一条线上全部加上 x 即可
    if(l==r) return;
    int m = (l+r)>>1;
```

```
    if(p<=m) modify(l, m, sgt[k].l, p, x);
    else modify(m+1, r, sgt[k].r, p, x);
}
void merge(int &x,int y)         // 把 y 子树的内容合并到 x 子树上，此写法不消耗空间
{
    if(!(x&&y)) x |= y;            // 如果二者有 NULL 结点
    else
    {
        sgt[x].val += sgt[y].val;    // 维护加法，直接加就是了
        merge(sgt[x].l, sgt[y].l);   // 递归合并两结点的左子树
        merge(sgt[x].r, sgt[y].r);   // 递归合并两结点的右子树
    }
}
int split(int l,int r,int &k,int x,int y)    // 从 k 子树中分离出 [x,y] 区间并返回新结点编号，空间复杂度 O(2logn)
{
    int n = ++cnt;
    if(x<=l&&y>=r)       // 如果 k 结点维护的区间在 [x,y] 中
    {
        sgt[n] = sgt[k];     // 直接拿过来便是
        k = 0;               // 置为 NULL，断掉联系
        return;
    }
    int m = (l+r)>>1;
    if(x<=m) sgt[n].l = split(l, m, sgt[k].l, x, y);
    if(y>m)  sgt[n].r = split(m+1, r, sgt[k].r, x, y);
    pushup(k); pushup(n);
    return n;
}
ll query(int l,int r,int k,int x,int y)
{
    if(x<=l&&y>=r) return sgt[k].val;
    int m = (l+r)>>1;
    ll sum = 0;
    if(x<=m) sum += query(l,m,sgt[k].l,x,y);
    if(y>m)  sum += query(m+1,r,sgt[k].r,x,y);
    return sum;
}
```

线段树分裂合并解决多次区间正反排序问题

```
struct Node { int ch[2],val; } sgt[70*maxn];
int cnt;
inline int& ls(int k) { return sgt[k].ch[0]; }
inline int& rs(int k) { return sgt[k].ch[1]; }
int modify(int l,int r,int p)       // 单点修改，固定修改 p 位置为 1
{
    int n = ++cnt;
    sgt[n].val = 1;
    if(l!=r)
    {
        int m = (l+r)>>1;
        if(p<=m) ls(n) = modify(l, m, p);
        else rs(n) = modify(m+1, r, p);
    }
    return n;
}
void merge(int &x, int y)           // 经典 merge，略
```

```
{
    if(!x||!y) x |= y;
    else
    {
        sgt[x].val += sgt[y].val;
        merge(ls(x), ls(y));
        merge(rs(x), rs(y));
    }
}
int split(int k, int kth, bool rev)      // 把前 kth 个数字之外的部分分裂出去并返回结点编号
{
    if(sgt[k].val==kth) return 0;      // 如果不够 kth 个无法分裂(不会出现 < 的情况所以 == 相当于 <=)
    int n = ++cnt;
    sgt[n].val = sgt[k].val-kth;         // 分裂出去的部分的数量为总数减 kth 个
    sgt[k].val = kth;                    // 剩下 kth 个
    int val = sgt[sgt[k].ch[rev]].val;  // 靠前子树值,如果正序那么就是判断左子树数字数量,如果倒序那么就是判断右子树数字数量
    if(val>=kth)                         // 如果在靠前子树
    {
        sgt[n].ch[rev] = split(sgt[k].ch[rev], kth, rev);    // 分裂靠前子树
        sgt[n].ch[!rev] = sgt[k].ch[!rev];       // 靠前子树只剩 kth 个,靠后子树自然是要归给新结点(新结点是剩余部分嘛)
        sgt[k].ch[!rev] = 0;                     // 与分出去的结点断开联系
    }
    else sgt[n].ch[!rev] = split(sgt[k].ch[!rev], kth - val, rev);  // 如果在靠后子树,直接分裂即可,因为靠前子树还是应该属于老树
    return n;
}
typedef tuple<int, bool, int> tp3;  // < 维护区间的左端点,是否倒序排序,根结点编号 >
// 只以存放的数字个数来去重,不写这个仿函数就会因为继续比较 tuple 的另两个元素而导致未去重
struct Cmp { bool operator() (const tp3& t1, const tp3& t2) const { return get<0>(t1) < get<0>(t2); } };
set<tp3, Cmp> rt;
void data(int l, int r, int k, vector<int>& v)  // 取序列
{
    if(!k) return;
    if(l==r) v.push_back(l);
    int m = (l+r)>>1;
    data(l, m, ls(k), v);
    data(m+1, r, rs(k), v);
}
vector<int> print(int l, int r)
{
    vector<int> ret;
    for(auto t : rt)
    {
        vector<int> v;
        data(l, r, get<2>(t), v);
        if(get<1>(t)) ret.insert(ret.end(), v.rbegin(), v.rend());
        else ret.insert(ret.end(), v.begin(), v.end());
    }
    return ret;
}
auto split(int p)   // 使得 rt 中存在以 p 为区间左端点的元素
{
    auto it = rt.lower_bound(tp3(p, false, 0));
    if(get<0>(*it)==p) return it;   // 满足条件
    it--;   // 否则 p 一定在前一个元素中
    int l, n; bool rev;
    tie(l, rev, n) = *it;
```

```
    // 给剩下 p-l 个元素，其余的分裂成新树，这样 p 就会是其所在线段树维护区间的左端点了
    return rt.insert(tp3(p, rev, split(n, p-l, rev))).first;
}
void solve(int l, int r, bool rev)
{
    if(l>r) return;
    auto itl = split(l), itr = split(r+1);
    int n = 0;
    for(auto it = itl; it!=itr; ++it) merge(n, get<2>(*it)); // 为使他们具有相同的排序顺序，合并起来再塞回去
    rt.erase(itl, itr);                         // 从 rt 中删去老信息
    rt.insert(tp3(l, rev, n));                  // 塞回去
}
int n,m;
cin>>n>>m;
for(int i=1;i<=n;i++)
{
    int p;
    cin>>p;
    rt.insert(tp3(i, 0, modify(1,n,p)));
}
while(m--)
{
    int opt,l,r;
    cin>>opt>>l>>r;
    solve(l,r,opt); // opt 为 0 是升序排序
}
for(auto i : print(1, n)) cout<<i<<' ';
```

带修主席树（树状数组套值域线段树）

```
const int maxn = 1e5+5;
vector<int> o;
int getid(int x) { return lower_bound(o.begin(), o.end(), x)-o.begin()+1; };
struct Node { int l,r,val; } sgt[maxn*400];     // O(nlog^2n)
int cnt, root[maxn];
int& ls(int k) { return sgt[k].l; }
int& rs(int k) { return sgt[k].r; }
void modify(int l,int r,int p,int x,int& k)
{
    if(!k) k = ++cnt;
    sgt[k].val += x;
    if(l==r) return;
    int m = (l+r)>>1;
    if(p<=m) modify(l,m,p,x,ls(k));
    else modify(m+1,r,p,x,rs(k));
}
int query(int l,int r,vector<int>& L,vector<int>& R,int k)  // 区间 k 小
{
    if(l==r) return l;
    int m = (l+r)>>1;
    int sum = 0;
    for(auto i : R) sum += sgt[ls(i)].val;
    for(auto i : L) sum -= sgt[ls(i)].val;
    if(k<=sum)
    {
        transform(L.begin(), L.end(), L.begin(), ls);
```

```
        transform(R.begin(), R.end(), R.begin(), ls);
        return query(l,m,L,R,k);
    }
    else
    {
        transform(L.begin(), L.end(), L.begin(), rs);
        transform(R.begin(), R.end(), R.begin(), rs);
        return query(m+1,r,L,R,k-sum);
    }
}
int n;
int lowbit(int x) { return x&-x; }
void modify(int p,int q,int x)
{
    for(int i=p;i<=n;i+=lowbit(i)) modify(1,o.size(),q,x,root[i]);
}
int query(int l,int r,int k)
{
    vector<int> L,R;
    for(int i=r;i;i-=lowbit(i)) R.push_back(root[i]);
    for(int i=l-1;i;i-=lowbit(i)) L.push_back(root[i]);
    return query(1,o.size(),L,R,k);
}

int m;
cin>>n>>m;
vector<int> v(n+1);
for(int i=1;i<=n;i++) cin>>v[i];
o.insert(o.end(), v.begin()+1, v.end());
vector<tuple<int,int,int>> w;
for(int i=0;i<m;i++)
{
    char opt;
    cin>>opt;
    if(opt=='Q')
    {
        int l,r,k;
        cin>>l>>r>>k;
        w.emplace_back(l,r,k);  // 查 [l,r] k 小
    }
    else
    {
        int p,x;
        cin>>p>>x;
        w.emplace_back(p,x,-1); // p 位置改为 x
        o.push_back(x);
    }
}
sort(o.begin(), o.end());
o.erase(unique(o.begin(), o.end()), o.end());
for(int i=1;i<=n;i++) modify(i, getid(v[i]), 1);
for(auto [x,y,z] : w)
{
    if(~z) cout<<o[query(x,y,z)-1]<<'\n';
    else
    {
```

```
        modify(x, getid(v[x]), -1);
        v[x] = y;
        modify(x, getid(v[x]), 1);
    }
}
```