带修改主席树

Idea：主席树不再维护前缀和，而是按树状数组的空间逻辑去维护。例如，第 4 颗主席树是第 2 颗主席树和第 4 个元素的“和”，第 6 颗主席树是 5, 6 两元素的“和”，每一次修改或查询都关联到对应的树状数组的 $\lg n$ 颗主席树。

ATT：空间开 400 倍。

Complexity：$O(n\lg^2 n)$

Code：

```cpp
#include<bits/stdc++.h>

using namespace std;

const int N = 200005;

int T, n, m, a[N], func[N], t[N], maxx;
struct Query{
    char ch[2];
    int x, y, k;
}que[N];

inline void disc(){
    sort(t+1, t+t[0]+1);
    int len = unique(t+1, t+t[0]+1) - (t+1);
    for(int i = 1; i <= n; i++){
        int d = lower_bound(t+1, t+len+1, a[i]) - t;
        func[d] = a[i], a[i] = d, maxx = max(maxx, d);
    }
    for(int i = 1; i <= m; i++){
        if(que[i].ch[0] == 'C'){
            int d = lower_bound(t+1, t+len+1, que[i].y) - t;
            func[d] = que[i].y, que[i].y = d, maxx = max(maxx, d);
        }
    }
}

struct segTree{
    int l, r, lson, rson, size;
}tr[N * 400];
int cnt, root[N], num1[N], num2[N];
inline void pushup(int id){
    tr[id].size = tr[tr[id].lson].size + tr[tr[id].rson].size;
}
void build(int id, int l, int r){
    tr[id].l = l, tr[id].r = r;
    if(l == r){
        tr[id].lson = tr[id].rson = tr[id].size = 0;
        return;
    }
    tr[id].lson = ++cnt, tr[id].rson = ++cnt;
    int mid = (l + r) >> 1;
    build(tr[id].lson, l, mid);
    build(tr[id].rson, mid+1, r);
    pushup(id);
}
void add(int cur, int l, int r, int pos, int val){
    if(l == r){
        tr[cur].size += val;
        return;
    }
    int mid = (l + r) >> 1;
    if(!tr[cur].lson)   tr[cur].lson = ++cnt;
    if(!tr[cur].rson)   tr[cur].rson = ++cnt;
    if(pos <= mid)    add(tr[cur].lson, l, mid, pos, val);
    else    add(tr[cur].rson, mid+1, r, pos, val);
    pushup(cur);
}

inline int lowbit(int x){ return x & -x; }
void add(int x, int pos, int val){
    // add (val) on the position(pos) of the (x)th tree (which is rooted with root[x])
    while(x <= n){
        if(!root[x])    root[x] = ++cnt; // if there's not a tree rooted with root[x], then build a new one
        add(root[x], 1, maxx, pos, val); // modify the tree (cuz we don't need to save the previous tree)
        x += lowbit(x);
    }
}
int queryKth(int l, int r, int k){
    if(l == r)  return l;
    int leftSize = 0;
    for(int i = 1; i <= num1[0]; i++)   leftSize -= tr[tr[num1[i]].lson].size;
    for(int i = 1; i <= num2[0]; i++)   leftSize += tr[tr[num2[i]].lson].size;
    int mid = (l + r) >> 1;
    if(k <= leftSize){
        // record the next indices we need to modify
        for(int i = 1; i <= num1[0]; i++)   num1[i] = tr[num1[i]].lson;
        for(int i = 1; i <= num2[0]; i++)   num2[i] = tr[num2[i]].lson;
        return queryKth(l, mid, k);
    }
    else{
        // record the next indices we need to modify
        for(int i = 1; i <= num1[0]; i++)   num1[i] = tr[num1[i]].rson;
        for(int i = 1; i <= num2[0]; i++)   num2[i] = tr[num2[i]].rson;
        return queryKth(mid+1, r, k - leftSize);
    }
}

int main(){
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i++) scanf("%d", &a[++t[0]]), t[i] = a[i];
    for(int i = 1; i <= m; i++){
        scanf("%s", que[i].ch);
        if(que[i].ch[0] == 'Q')   scanf("%d%d%d", &que[i].x, &que[i].y, &que[i].k);
        else    scanf("%d%d", &que[i].x, &que[i].y), t[++t[0]] = que[i].y;
    }
    disc();
    build(root[0] = 0, 1, maxx); // root[0] = 0 --- build an empty tree
    for(int i = 1; i <= n; i++) add(i, a[i], 1);
    for(int i = 1; i <= m; i++){
        if(que[i].ch[0] == 'Q'){
            num1[0] = num2[0] = 0;
            // record the root we need to modify
            int x = que[i].x - 1;   while(x){ num1[++num1[0]] = root[x]; x -= lowbit(x); }
            x = que[i].y;           while(x){ num2[++num2[0]] = root[x]; x -= lowbit(x); }
            printf("%d\n", func[queryKth(1, maxx, que[i].k)]);
        }
        else{
            add(que[i].x, a[que[i].x], -1);
            add(que[i].x, que[i].y, 1);
            a[que[i].x] = que[i].y;
        }
    }
    return 0;
}
```