南京工業大學
NANJING TECH UNIVERSITY

ICPC Template Manual

作者: 贺梦杰

September 20, 2019

Contents

Chapter 1

基础

Chapter 2

搜索

Chapter 3

动态规划

Chapter 4

字符串

Chapter 5

数据结构

5.1 树状数组套权值线段树

树状数组套权值线段树通常用来解决一种二维查询，第一维是区间，第二维是值。
最典型的例子就是带修改的区间 k 小值查询，思路是，把二分答案的操作和查询小于一个值的数的数量两种操作结合起来。
在修改操作进行时，先在线段树上从上往下跳到被修改的点，删除所经过的点所指向的动态开点权值线段树上的原来的值，然后插入新的值。
在查询答案时，先取出该区间覆盖在线段树上的所有点，然后用类似于静态区间 k 小值的方法，将这些点一起向左儿子或向右儿子跳。如果所有这些点左儿子存储的值大于等于 k，则往左跳，否则往右跳。

```
/*
例题：洛谷 P2617 Dynamic Rankings
线段树套动态开点权值线段树
*/

#include <bits/stdc++.h>

using namespace std;
const int N = 2e5 + 10;  // FIXME:

struct Qr {
    char op[3];   // 哪种操作
    int l, r, k;  // 查询
    int p, v;     // 修改
} qr[N];

int n, m, a[N];    // 元素个数、操作个数、原数组
int q1[N], q2[N];  // 待查区间左右端点的前缀和，数组元素表示组成前缀和的权值线段树的根结点编号
int cnt1, cnt2;    // 两个前缀和被划分成的节点个数

// 离散化
int i2x[N], len;
inline int x2i(int x) {
    return lower_bound(i2x + 1, i2x + 1 + len, x) - i2x;
}

struct SegT {
    int cnt;                        // 已经使用的结点个数
    int rt[N * 2];                  // rt[i]：树状数组中下标为i的节点对应的权值线段树的根结点编号
    int lc[N * 300], rc[N * 300];  // 左右儿子编号
    int s[N * 300];                 // 结点中值的个数

    // o是当前结点的编号，[l,r]表示当前结点所表示的区间，v是要添加或删除的权值，d表示添加或删除
    void update(int& o, int l, int r, int v, int d) {
        if (!o)
            o = ++cnt;
        s[o] += d;
        if (l == r)
            return;
        int m = (l + r) / 2;
        if (v <= m)
            update(lc[o], l, m, v, d);
        else
            update(rc[o], m + 1, r, v, d);
    }
} st;

inline int lowbit(int x) {
    return -x & x;
}

// p是要修改的位置，v是要修改的权值，d=-1或1表示删除或添加
void upd(int p, int v, int d) {
    for (; p <= n; p += lowbit(p))
        st.update(st.rt[p], 1, len, v, d);
}
```

```
57
58  // 获取构成区间[1,p]的所有权值线段树的根结点，放到a[]中，共cnt个
59  void gtv(int p, int a[], int& cnt) {
60      cnt = 0;
61      for (; p; p -= lowbit(p))
62          a[++cnt] = st.rt[p];
63  }
64
65  // [l,r]表示当前两组结点表示的值域，查询第k小值，返回的是离散化之后的值
66  int qry(int l, int r, int k) {
67      if (l == r)
68          return l;
69
70      int m = (l + r) / 2, ltot = 0, i;
71
72      // 统计左子树中权值个数，前缀和之差
73      for (i = 1; i <= cnt2; i++)
74          ltot += st.s[st.lc[q2[i]]];
75      for (i = 1; i <= cnt1; i++)
76          ltot -= st.s[st.lc[q1[i]]];
77
78      if (ltot >= k) {  // 向左搜
79          for (i = 1; i <= cnt2; i++)
80              q2[i] = st.lc[q2[i]];
81          for (i = 1; i <= cnt1; i++)
82              q1[i] = st.lc[q1[i]];
83          return qry(l, m, k);
84      } else {  // 向右搜
85          for (i = 1; i <= cnt2; i++)
86              q2[i] = st.rc[q2[i]];
87          for (i = 1; i <= cnt1; i++)
88              q1[i] = st.rc[q1[i]];
89          return qry(m + 1, r, k - ltot);
90      }
91  }
92
93  int main() {
94      int i;
95
96      scanf("%d%d", &n, &m);
97      for (i = 1; i <= n; i++)
98          scanf("%d", &a[i]), i2x[++len] = a[i];
99
100     for (i = 1; i <= m; i++) {
101         scanf("%s", qr[i].op);
102         if (qr[i].op[0] == 'Q')
103             scanf("%d%d%d", &qr[i].l, &qr[i].r, &qr[i].k);
104         else
105             scanf("%d%d", &qr[i].p, &qr[i].v), i2x[++len] = qr[i].v;
106     }
107
108     // 离散化
109     sort(i2x + 1, i2x + 1 + len);
110     len = unique(i2x + 1, i2x + 1 + len) - i2x - 1;
111
112     // 初始化树状数组
113     for (i = 1; i <= n; i++)
114         upd(i, x2i(a[i]), 1);
115
116     // 操作
117     for (i = 1; i <= m; i++) {
118         if (qr[i].op[0] == 'C') {                // 修改
119             upd(qr[i].p, x2i(a[qr[i].p]), -1);  // 删去旧值
120             upd(qr[i].p, x2i(qr[i].v), 1);      // 添加新值
121             a[qr[i].p] = qr[i].v;
122         } else {  // 查询
```

```
123             gtv(qr[i].l - 1, q1, cnt1);
124             gtv(qr[i].r, q2, cnt2);
125             printf("%d\n", i2x[qry(1, len, qr[i].k)]);
126         }
127     }
128
129     return 0;
130 }
```

Chapter 6

图论