**主席树--动态区间第k小**

模板题在这里洛谷2617。

**先对几个问题做一个总结：**

阅读本文需要有主席树的基础，也就是通过区间kth的模板题。

**静态整体kth：**

sort一下找第k小，时间复杂度O(nlogn)。

**动态整体kth：**

权值线段树维护一下，时间复杂度O(nlogn)。

**静态区间kth：**

主席树维护，时间复杂度O(nlogn)。

**动态区间kth：**

就是本次的标题。

回忆一下主席树是如何维护静态区间kth的。

建立可持久化线段树后，利用前缀和的思想查询区间的kth。

所以我们想对区间kth带修改操作，前缀和是关键。

我们在维护普通前缀和，支持查询和修改操作时，用的是什么数据结构呢？

* 树状数组/线段树。

所以这时候我们大致有一个概念了，动态主席树和主席树在**数据结构**上已经多少有点不一样了。

* **动态主席树：树套树**。
* **静态主席树：可持久化权值线段树**。

怎么套是一个问题，但简单想想可以发现，我们在**外层维护一颗树状数组，树状数组的每个节点（内层）维护权值线段树（的根节点）**，可以解决这个问题。

* 修操作：
  + 如果将一个位置的数字a\_i=x修改为y，那么在外层树状数组上，我们需要修改logn个节点，同时对于每个节点（代表了一颗权值线段树），分别有logn个节点受影响，所以修改复杂度为O((logn)^2)。
* 查操作：
  + 对于每次[L,R]之间查询，我们先提取这区间的logn个根节点，然后将问题转换为静态主席树求区间k小。
  + 时间复杂度为O((logn)^2)。

所以总的算下来时间复杂度在O(n(logn)^2)上。

接下来解决一下空间的问题。我们知道线段树的空间复杂度是O(4n)  
的，也就是O(n)，树状数组的复杂度为O(n)，那么如此算下来空间复杂度达到了O(n^2)。

思考一下怎么优化？

我们刚刚计算的空间复杂度，基于对每个节点都把完整的权值线段树开出来。

我们查询/修改操作的规模是(logn)^2级别的。

那是不是可以动态开点。

对于我们能访问到的节点创立节点，对于不能访问到的，就不管了。

附luogu2617  
代码。

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 | #include<bits/stdc++.h> using namespace std; const int maxn = 1e5+10; int n, m, a[maxn], b[maxn<<1], len;  struct Query {     int op;     int i, j, k; }q[maxn];  //1e5的log大概在20左右 20\*20=400 int sum[maxn\*400]; int ls[maxn\*400]; int rs[maxn\*400]; int rt[maxn\*400]; int tot;  void update\_SgT(int &rt, int l, int r, int x, int val) {     if(!rt) rt = ++tot;     if(l == r)     {         sum[rt] += val;         return;     }     int mid = (l+r) >> 1;     if(x <= mid) update\_SgT(ls[rt], l, mid, x, val);     else update\_SgT(rs[rt], mid+1, r, x, val);     sum[rt] = sum[ls[rt]] + sum[rs[rt]]; }  inline int lowbit(int x){     return x&(-x); }  void update\_BIT(int pos, int x, int val) {     for(int i = pos; i <= n; i += lowbit(i))         update\_SgT(rt[i], 1, len, x, val); }  ///提取区间线段树的根节点 int rt1[maxn], rt2[maxn], cnt1, cnt2; void locate(int l, int r) {     cnt1 = cnt2 = 0;     for(int i = l-1; i; i -= lowbit(i))         rt1[++cnt1] = rt[i];     for(int i = r; i; i -= lowbit(i))         rt2[++cnt2] = rt[i]; }  int ask(int l, int r, int k) {     if(l == r) return l;     int mid = (l+r) >> 1;     int suml = 0;     for(int i = 1; i <= cnt1; i++)         suml -= sum[ls[rt1[i]]];     for(int i = 1; i <= cnt2; i++)         suml += sum[ls[rt2[i]]];     if(suml >= k)     {         for(int i = 1; i <= cnt1; i++)             rt1[i] = ls[rt1[i]];         for(int i = 1; i <= cnt2; i++)             rt2[i] = ls[rt2[i]];         return ask(l, mid, k);     }     else     {         for(int i = 1; i <= cnt1; i++)             rt1[i] = rs[rt1[i]];         for(int i = 1; i <= cnt2; i++)             rt2[i] = rs[rt2[i]];         return ask(mid+1, r, k-suml);     }  }   int main() {     scanf("%d%d", &n, &m);     for(int i = 1; i <= n; i++)     {         scanf("%d", &a[i]);         b[++len] = a[i];     }      char op[2];     for(int i = 1; i <= m; i++)     {         scanf("%s", op);         if(op[0] == 'Q')         {             q[i].op = 0;             scanf("%d%d%d", &q[i].i, &q[i].j, &q[i].k);         }         else         {             q[i].op = 1;             scanf("%d%d", &q[i].i, &q[i].k);             b[++len] = q[i].k;         }     }      //数值离散化     sort(b+1, b+1+len);     len = unique(b+1, b+1+len)-b-1;     for(int i = 1; i <= n; i++)         a[i] = lower\_bound(b+1, b+len+1, a[i])-b;     for(int i = 1; i <= m; i++)         if(q[i].op) q[i].k = lower\_bound(b+1, b+len+1, q[i].k)-b;      //建树(动态开点形式)     for(int i = 1; i <= n; i++)         update\_BIT(i, a[i], 1);      for(int i = 1; i <= m; i++)     {         if(q[i].op)         {             update\_BIT(q[i].i, a[q[i].i], -1);             a[q[i].i] = q[i].k;             update\_BIT(q[i].i, q[i].k, 1);         }         else         {             locate(q[i].i, q[i].j);             int ans = b[ask(1, len, q[i].k)];             printf("%d ", ans);         }     }      return 0; } |