

## 249. 统计前面比自己小的数的个数

给定一个整数数组（下标由 0 到 n-1，n 表示数组的规模，取值范围由 0 到10000）。对于数组中的每个 `ai` 元素，请计算 `ai` 前的数中比它小的元素的数量。

We suggest you finish problem [Segment Tree Build](#), [Segment Tree Query II](#) and [Count of Smaller Number](#) first.

样例:对于数组 `[1,2,7,8,5]`，返回 `[0,1,2,3,2]`

思路：

1. 如何找到问题的切入点，对于每一个元素`A[i]`我们查询比它小数，转换成区间的查询就是查询在它前面的数当中有多少在区间`[0, A[i] - 1]`当中。
2. 因此我们可以为`0-10000`区间建树，并将所有区间count设为0。每一个最小区间（即叶节点）的count代表到目前为止该数的数量。  
然后开始遍历数组，遇到`A[i]`时，去查`0 ~ A[i]-1`区间的count即这个区间中有多少数存在，这就是比`A[i]`小的数的数量。查完后将`A[i]`区间的count加1即可，也就是把`A[i]`插入到线段树 `i` 的位置上。
3. 具体举例：`A = [1,2,7,8,5]`，最大值为8. 我们得到一棵空的SegmentTree，对应的前9个位置的区间是`[0, 0, 0, 0, 0, 0, 0, 0, 0]`。即该辅助数组第5个元素代表数组中5出现的次数。`ans[A[i]]`记录`A[i]`到目前为止出现的次数。

`A[0] = 1`，查区间`[0, 0]`的和，得到0，然后在1的地方加1，得到 `[0, 1, 0, 0, 0, 0, 0, 0, 0]`

`A[1] = 2`，查区间`[0, 1]`的和，得到1，然后在2的地方加1，得到 `[0, 1, 1, 0, 0, 0, 0, 0, 0]`

`A[2] = 7`，查区间`[0, 6]`的和，得到2，然后在7的地方加1，得到 `[0, 1, 1, 0, 0, 0, 0, 1, 0]`

`A[3] = 8`，查区间`[0, 7]`的和，得到3，然后在8的地方加1，得到 `[0, 1, 1, 0, 0, 0, 0, 1, 1]`

`A[4] = 5`，查区间`[0, 4]`的和，得到2，然后在5的地方加1，得到 `[0, 1, 1, 0, 0, 1, 0, 1, 1]`

所以我们得到结果序列为：`[0, 1, 2, 3, 2]`

总结：本题用到了区间单点修改和区间和查询两个主要的线段树操作。

```
1 public class Solution {
2     /**
3      * @param A: an integer array
4      * @return: A list of integers includes the index of the first number and the
5      *         index of the last number
6      */
7     public List<Integer> countOfSmallerNumberII(int[] A) {
8         // write your code here
9         root = build(0, 10000);
10        ArrayList<Integer> ans = new ArrayList<Integer>();
11        int res;
12        for( int i = 0; i < A.length; i++){
13            res = 0;
14            if( A[i] > 0 ){
15                res = query(root, 0, A[i] - 1);
16            }
17            modify(root, A[i], 1);
18            ans.add(res);
19        }
20        return ans;
21    }
22 }
```

```

21
22 class SegmentTreeNode{
23     public int start, end;
24     public int count;
25     public SegmentTreeNode left, right;
26     public SegmentTreeNode(int start, int end, int count){
27         this.start = start;
28         this.end = end;
29         this.count = count;
30         this.left = this.right = null;
31     }
32 }
33 SegmentTreeNode root;
34 public SegmentTreeNode build(int start, int end){
35     if( start > end )
36         return null;
37     SegmentTreeNode node = new SegmentTreeNode(start, end, 0);
38     if( start != end ){
39         int mid = (start + end)/2;
40         node.left = build(start, mid);
41         node.right = build(mid + 1, end);
42     }
43     return node;
44 }
45 public int query(SegmentTreeNode root, int start, int end){
46     if( start == root.start && end == root.end )
47         return root.count;
48     int mid = (root.start + root.end)/2;
49     int leftCount = 0, rightCount = 0;
50     if( start <= mid ){
51         leftCount = query( root.left, start, Math.min(end, mid) );
52     }
53     if( mid < end ){
54         rightCount = query( root.right, Math.max(mid+1, start), end );
55     }
56     return leftCount + rightCount;
57 }
58 public void modify(SegmentTreeNode root, int index, int value){
59     if( root.start == index && root.end == index ){
60         root.count += value;
61         return;
62     }
63     int mid = (root.start + root.end)/2;
64     if( index <= mid )
65         modify(root.left, index, value);
66     else
67         modify(root.right, index, value);
68     root.count = root.left.count + root.right.count;
69 }
70 }

```