6894. Sum of Imbalance Numbers of All Subarrays

 $My\ Submissions\ (/contest/weekly-contest-352/problems/sum-of-imbalance-numbers-of-all-subarrays/submissions/)$

Back to Contest (/contest/weekly-contest-352/)

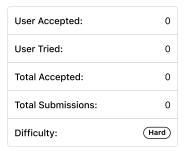
The **imbalance number** of a **0-indexed** integer array arr of length n is defined as the number of indices in sarr = sorted(arr) such that:

- $0 \le i \le n 1$, and
- sarr[i+1] sarr[i] > 1

Here, sorted(arr) is the function that returns the sorted version of arr.

Given a **0-indexed** integer array nums , return the sum of imbalance numbers of all its subarrays.

A **subarray** is a contiguous **non-empty** sequence of elements within an array.



Example 1:

```
Input: nums = [2,3,1,4]
Output: 3
Explanation: There are 3 subarrays with non-zero imbalance numbers:
- Subarray [3, 1] with an imbalance number of 1.
- Subarray [3, 1, 4] with an imbalance number of 1.
- Subarray [1, 4] with an imbalance number of 1.
The imbalance number of all other subarrays is 0. Hence, the sum of imbalance numbers of all the subarrays of nums is 3.
```

Example 2:

```
Input: nums = [1,3,3,3,5]
Output: 8
Explanation: There are 7 subarrays with non-zero imbalance numbers:
- Subarray [1, 3] with an imbalance number of 1.
- Subarray [1, 3, 3] with an imbalance number of 1.
- Subarray [1, 3, 3, 3] with an imbalance number of 1.
- Subarray [1, 3, 3, 3, 5] with an imbalance number of 2.
- Subarray [3, 3, 3, 5] with an imbalance number of 1.
- Subarray [3, 3, 5] with an imbalance number of 1.
- Subarray [3, 5] with an imbalance number of 1.
The imbalance number of all other subarrays is 0. Hence, the sum of imbalance numbers of all the subarrays of nums is 8.
```

Constraints:

- 1 <= nums.length <= 1000
- 1 <= nums[i] <= nums.length

```
JavaScript
                                                                                                                       क
                                                                                                                             \mathfrak{C}
                                                                                                                                    Φ
 1 v class SplayNode {
 2 ▼
         constructor(value) {
 3
             this.parent = null;
 4
             this.left = null;
 5
             this.right = null;
 6
             this.val = value;
 7
             this.sum = value;
 8
             this.sz = 1;
 9
        }
10 •
        update() {
             this.sz = (this.left != null ? this.left.sz : 0) + (this.right != null ? this.right.sz : 0) + 1;
11
             this.sum = (this.left != null ? this.left.sum : 0) + (this.right != null ? this.right.sum : 0) + this.val;
12
13
14
         isLeft() {
             return this.parent != null && this.parent.left == this;
15
16
17 •
         isRight() {
18
             return this.parent != null && this.parent.right == this;
```

```
19
20 ▼
        isRoot(guard = null) {
21
             return this.parent == guard;
22
23
   }
24
25
   // MultiSet
26 v class SplayTree {
27 ▼
        constructor() {
28
            this.root = null;
29
             this.cmp = (x, y) \Rightarrow x >= y ? 0 : 1;
30
31 ▼
        zig(x) { // right rotation
             let y = x.parent;
32
             if (x.right != null) x.right.parent = y;
33
34
            y.left = x.right;
35
            x.right = y;
36 ▼
            if (y.isLeft()) {
                 y.parent.left = x;
37
38 ▼
             } else if (y.isRight()) {
39
                 y.parent.right = x;
40
41
            x.parent = y.parent;
            y.parent = x;
42
43
            y.update();
44
            x.update();
45
        }
        zag(x) { // left rotation
46 ▼
47
             let y = x.parent;
48
             if (x.left != null) x.left.parent = y;
49
            y.right = x.left;
50
            x.left = y;
51 ▼
             if (y.isLeft()) {
                 y.parent.left = x;
52
53 ▼
            } else if (y.isRight()) {
54
                 y.parent.right = x;
55
56
            x.parent = y.parent;
57
            y.parent = x;
58
            y.update();
59
            x.update();
60
        zigzig(x) { // RR
61 •
62
            this.zig(x.parent);
63
             this.zig(x);
64
65
        zigzag(x) { // RL}
66
             this.zig(x);
             this.zag(x);
67
68
69 •
        zagzag(x) { // LL
70
             this.zag(x.parent);
71
             this.zag(x);
72
        }
73 ▼
        zagzig(x) { // LR
74
             this.zag(x);
75
             this.zig(x);
76
        splay(node, guard = null) { // splay a "node" just under a "guard", which is default to splay to the "root".
77 ▼
78 ▼
             while (!node.isRoot(guard)) {
79 ▼
                 if (node.parent.isRoot(guard)) {
80 ▼
                     if (node.isLeft()) {
81
                         this.zig(node);
82 ▼
                     } else {
83
                         this.zag(node);
84
85 ▼
                 } else {
                     if (node.parent.isLeft()) {
86 ▼
87 ▼
                         if (node.isLeft()) {
88
                              this.zigzig(node);
89 •
                         } else {
90
                              this.zagzig(node);
91
                     } else {
92 ▼
93 ▼
                         if (node.isRight()) {
94
                              this.zagzag(node);
                         } else {
```

```
96
                              this.zigzag(node);
97
                         }
98
                     }
99
                 }
100
101
             if (quard == null) this.root = node; // reset "root" to "node".
102
103
         LastNode(x) {
104
             this.splay(x);
105
             let node = x.left;
106
             if (node == null) return null;
107
             while (node.right != null) node = node.right;
108
             this.splay(node);
109
             return node;
110
         NextNode(x) {
111
112
             this.splay(x);
             let node = x.right;
113
114
             if (node == null) return null;
115
             while (node.left != null) node = node.left;
116
             this.splay(node);
117
             return node;
         }
118
119 •
         find(value) {
120
             return this.findFirstOf(value);
121
122 🔻
         findFirstOf(value) {
             let node = this.root, res = null, last_visited = null;
123
124 ▼
             while (node != null) {
125
                 last_visited = node;
                 if (this.cmp(value, node.val)) {
126 •
127
                     node = node.left;
                 } else if (this.cmp(node.val, value)) {
128 •
129
                     node = node.right;
130 ▼
                 } else {
131
                     res = node;
132
                     node = node.left;
                 }
133
134
             if (last_visited != null) this.splay(last_visited);
135
136
             return res;
137
         findLastOf(value) {
138 •
139
             let node = this.root, res = null, last_visited = null;
140 ▼
             while (node != null) {
141
                 last_visited = node;
142
                 if (this.cmp(value, node.val)) {
143
                      node = node.left;
                 } else if (this.cmp(node.val, value)) {
144
145
                      node = node.right;
                 } else {
146
147
                     res = node;
148
                     node = node.right;
149
                 }
150
             if (last_visited != null) this.splay(last_visited);
151
152
             return res;
153
154
         findRankOf(node) {
155
             this.splay(node);
156
             return node.left == null ? 0 : node.left.sz;
157
         findSuccessorOf(value) {
158
159
             let node = this.root, res = null, last_visited = null;
160 •
             while (node != null) {
161
                 last_visited = node;
162
                 if (this.cmp(value, node.val)) {
163
                      res = node;
                     node = node.left;
164
165
                 } else {
166
                     node = node.right;
167
168
169
             if (last_visited != null) this.splay(last_visited);
170
             return res;
171
         findPrecursorOf(value) {
```

```
173
             let node = this.root, res = null, last_visited = null;
174 ▼
             while (node != null) {
175
                 last_visited = node;
                 if (this.cmp(node.val, value)) {
176 ▼
177
                      res = node;
178
                     node = node.right;
179
                 } else {
                      node = node.left;
180
181
                 }
182
             if (last_visited != null) this.splay(last_visited);
183
184
185
         findKthNode(rank) {
186 •
             if (rank < 0 || rank >= this.size()) return null;
187
             let node = this.root;
188
189
             while (node != null) {
                 let leftsize = node.left == null ? 0 : node.left.sz;
190
191
                 if (leftsize == rank) break;
192 ▼
                 if (leftsize > rank) {
193
                     node = node.left;
194
                 } else {
195
                     rank -= leftsize + 1;
196
                     node = node.right;
                 }
197
198
             }
199
             this.splay(node);
200
             return node;
201
         }
202 •
         make(value) {
203
             return new SplayNode(value);
204
         }
205 •
         removeNode(node) {
206
             node = null;
207
208
209
                         ----- Public Usage ------
         insert(value) { // allow duplicates, tree nodes allow same value O(logN)
210 •
211 ▼
             if (this.root == null) {
212
                 this.root = this.make(value);
                 return this.root;
213
214
             let node = this.root;
215
216 •
             while (node != null) {
217 ▼
                 if (this.cmp(value, node.val)) {
218 ▼
                      if (node.left == null) {
219
                          node.left = this.make(value);
                          node.left.parent = node;
220
221
                         node = node.left;
                          break;
222
223
                     }
224
                     node = node.left;
225 •
                 } else {
226 •
                     if (node.right == null) {
227
                          node.right = this.make(value);
228
                          node.right.parent = node;
229
                          node = node.right;
230
                          break;
231
232
                     node = node.right;
233
                 }
234
235
             this.splay(node);
236
             return node;
237
         remove(value) { // remove one node, not remove all O(logN)
238
239
             let node = this.find(value);
             if (node == null) return false;
240
             this.splay(node);
241
242 •
             if (node.left == null) {
243
                 this.root = node.right;
244
                 if (node.right != null) node.right.parent = null;
245
                 this.removeNode(node);
246
                 return true;
247
             if (node.right == null) {
248 •
                 this.root = node.left;
```

```
250
                  if (node.left != null) node.left.parent = null;
251
                  this.removeNode(node);
252
                  return true;
253
254
             let last_node = this.LastNode(node);
255
             let next_node = this.NextNode(node);
256
             this.splay(last_node);
             this.splay(next_node, last_node);
257
258
             this.removeNode(next_node.left);
259
             next_node.left = null;
260
             next_node.update();
261
             last_node.update();
             return true;
262
263
         }
         has(value) { // O(logN)
264
265
             return this.count(value) > 0;
266
         count(value) { // O(logN)
267 ▼
             let x = this.findFirstOf(value);
268
269
             if (x == null) return 0;
             let rank_x = this.findRankOf(x);
270
271
             let y = this.findLastOf(value);
272
             let rank_y = this.findRankOf(y);
273
             return rank_y - rank_x + 1;
274
         }
275 •
         rankOf(value) { // The number of elements strictly less than value O(logN)
276
             let x = this.findPrecursorOf(value);
             return x == null ? 0 : this.findRankOf(x) + 1;
277
278
279 •
         findKth(rank) { // (0-indexed) 0(logN)
280
             let x = this.findKthNode(rank);
281
             return x == null ? null : (x.val);
282
         higher(value) { // > upper_bound 0(logN)
283 •
284
             let node = this.findSuccessorOf(value);
285
             return node == null ? null : (node.val);
286
         lower(value) { // < O(logN)}
287 🕶
             let node = this.findPrecursorOf(value);
288
289
             return node == null ? null : (node.val);
290
291 •
         first() {
292
             return this.findKth(0);
293
294 ▼
         last() {
295
             return this.findKth(this.size() - 1);
296
         }
         poll() {
297 •
             let res = this.first();
298
299
             this.remove(res);
300
             return res;
301
         }
         pollLast() {
302 •
303
             let res = this.last();
304
             this.remove(res);
305
             return res;
306
         }
307
         size() {
308
             return this.root == null ? 0 : this.root.sz;
309
310 ▼
         isEmpty() {
             return this.root == null;
311
312
313 ▼
         show() { // Get sorted values in the splay tree O(n).
314
             let res = [];
315 ▼
             const dfs = (x) \Rightarrow \{
                  if (x == null) return;
316
317
                  dfs(x.left);
                  res.push(x.val);
318
319
                  dfs(x.right);
320
             };
321
             dfs(this.root);
322
             return res;
323
         }
324
     }
325
326 v const sumImbalanceNumbers = (a) ⇒ {
```

```
327
         let n = a.length, res = 0;
328 ▼
         for (let i = 0; i < n; i++) {
             let tree = new SplayTree(), cnt = -1;
329
330 ▼
             for (let j = i; j < n; j++) {
331
                 let x = a[j];
                 if (!tree.has(x)) {
332 ▼
333
                      tree.insert(x);
334
                     cnt++;
335
                     if (tree.has(x - 1)) cnt--;
                     if (tree.has(x + 1)) cnt--;
336
337
                 }
338
                     += cnt;
339
             }
340
         }
341
         return res;
342
    };
```

 $\ \square$ Custom Testcase

Use Example Testcases

Submission Result: Accepted (/submissions/detail/984304909/) 3

More Details > (/submissions/detail/984304909/)

Run

△ Submit

Share your acceptance!

Copyright © 2023 LeetCode

Help Center (/support) | Jobs (/jobs) | Bug Bounty (/bugbounty) | Online Interview (/interview/) | Students (/student) | Terms (/terms) | Privacy Policy (/privacy)

United States (/region)