

onumber-of-achievabletransfer-requests/)

6911. Continuous Subarrays

My Submissions (/contest/weekly-contest-352/problems/continuous-subarrays/submissions/) Back to Contest (/contest/weekly-contest-352/) You are given a **0-indexed** integer array nums . A subarray of nums is called **continuous** if: User Accepted: 0 • Let i, i + 1, ..., j be the indices in the subarray. Then, for each pair of indices i <= i_1 , i_2 <= j, 0 <= User Tried: 0 $|nums[i_1] - nums[i_2]| \le 2$. Return the total number of **continuous** subarrays. Total Accepted: 0 A subarray is a contiguous **non-empty** sequence of elements within an array. **Total Submissions:** 0 Difficulty: Medium

Example 1:

```
Input: nums = [5,4,2,4]
Output: 8
Explanation:
Continuous subarray of size 1: [5], [4], [2], [4].
Continuous subarray of size 2: [5,4], [4,2], [2,4].
Continuous subarray of size 3: [4,2,4].
Thereare no subarrys of size 4.
Total continuous subarrays = 4 + 3 + 1 = 8.
It can be shown that there are no more continuous subarrays.
```

Example 2:

```
Input: nums = [1,2,3]
Output: 6
Explanation:
Continuous subarray of size 1: [1], [2], [3].
Continuous subarray of size 2: [1,2], [2,3].
Continuous subarray of size 3: [1,2,3].
Total continuous subarrays = 3 + 2 + 1 = 6.
```

Constraints:

- 1 <= nums.length <= 10⁵
- $1 \le nums[i] \le 10^9$

```
JavaScript
                                                                                                                             \mathfrak{C}
 1 v class SplayNode {
         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 ▼
             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
         isLeft() {
14 •
15
             return this.parent != null && this.parent.left == this;
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 {
        constructor() {
27 ▼
28
             this.root = null;
29
             this.cmp = (x, y) \Rightarrow x >= y ? 0 : 1;
30
31 ▼
        zig(x) { // right rotation
32
             let y = x.parent;
             if (x.right != null) x.right.parent = y;
33
34
            y.left = x.right;
35
             x.right = y;
36 ▼
             if (y.isLeft()) {
37
                 y.parent.left = x;
38 ▼
            } else if (y.isRight()) {
39
                 y.parent.right = x;
40
41
            x.parent = y.parent;
42
            y.parent = x;
43
            y.update();
44
            x.update();
45
        zag(x) { // left rotation
46 ▼
47
             let y = x.parent;
             if (x.left != null) x.left.parent = y;
48
49
            y.right = x.left;
            x.left = y;
50
             if (y.isLeft()) {
51 ▼
52
                 y.parent.left = x;
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);
67
             this.zag(x);
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
77 ▼
        splay(node, guard = null) { // splay a "node" just under a "guard", which is default to splay to the "root".
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 {
                        (node.parent.isLeft()) {
86 ▼
                     if
87 ▼
                         if (node.isLeft()) {
88
                              this.zigzig(node);
89 •
                         } else {
90
                              this.zagzig(node);
91
92 ▼
                     } else {
93 ▼
                         if (node.isRight()) {
94
                              this.zagzag(node);
95 •
                         } else {
96
                              this.zigzag(node);
97
98
                     }
99
                 }
```

```
101
             if (guard == null) this.root = node; // reset "root" to "node".
102
103
         LastNode(x) {
104
             this.splay(x);
105
             let node = x.left;
             if (node == null) return null;
106
107
             while (node.right != null) node = node.right;
108
             this.splay(node);
109
             return node;
110
111
         NextNode(x) {
112
             this.splay(x);
113
             let node = x.right;
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) {
123
             let node = this.root, res = null, last_visited = null;
             while (node != null) {
124 ▼
125
                 last_visited = node;
                 if (this.cmp(value, node.val)) {
126 •
127
                      node = node.left;
                 } else if (this.cmp(node.val, value)) {
128 •
                      node = node.right;
129
130 •
                 } else {
131
                     res = node;
132
                      node = node.left;
                 }
133
134
             }
135
             if (last_visited != null) this.splay(last_visited);
136
             return res;
137
         findLastOf(value) {
138 •
139
             let node = this.root, res = null, last_visited = null;
140 ▼
             while (node != null) {
                 last_visited = node;
141
142
                 if (this.cmp(value, node.val)) {
                      node = node.left;
143
144 v
                 } else if (this.cmp(node.val, value)) {
145
                      node = node.right;
146
                 } else {
147
                      res = node;
148
                      node = node.right;
149
                 }
150
             if (last_visited != null) this.splay(last_visited);
151
152
             return res;
153
         findRankOf(node) {
154 ▼
155
             this.splay(node);
             return node.left == null ? 0 : node.left.sz;
156
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;
164
                      node = node.left;
165 •
                 } else {
166
                     node = node.right;
167
                 }
168
             if (last_visited != null) this.splay(last_visited);
169
170
             return res;
171
172 ▼
         findPrecursorOf(value) {
173
             let node = this.root, res = null, last_visited = null;
174 ▼
             while (node != null) {
175
                 last_visited = node;
176 •
                 if (this.cmp(node.val, value)) {
177
                      res = node;
```

```
178
                     node = node.right;
179 •
                 } else {
180
                    node = node.left;
181
182
183
             if (last_visited != null) this.splay(last_visited);
184
             return res;
185
186 •
         findKthNode(rank) {
187
             if (rank < 0 || rank >= this.size()) return null;
             let node = this.root;
188
189
             while (node != null) {
                 let leftsize = node.left == null ? 0 : node.left.sz;
190
                 if (leftsize == rank) break;
191
192 •
                 if (leftsize > rank) {
193
                     node = node.left;
194
                 } else {
                     rank -= leftsize + 1;
195
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
         // ------
         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)) {
                     if (node.left == null) {
218 ▼
219
                         node.left = this.make(value);
                         node.left.parent = node;
220
                         node = node.left;
221
222
                         break;
223
224
                     node = node.left;
225 •
                 } else {
226 •
                     if (node.right == null) {
                         node.right = this.make(value);
227
228
                         node.right.parent = node;
229
                         node = node.right;
230
                         break;
231
                    }
232
                     node = node.right;
233
                 }
234
235
             this.splay(node);
             return node;
236
237
238
         remove(value) { // remove one node, not remove all O(logN)
239
             let node = this.find(value);
             if (node == null) return false;
240
241
             this.splay(node);
242 •
             if (node.left == null) {
243
                 this.root = node.right;
244
                 if (node.right != null) node.right.parent = null;
245
                 this.removeNode(node);
                 return true;
246
247
             if (node.right == null) {
248 v
249
                 this.root = node.left;
250
                 if (node.left != null) node.left.parent = null;
251
                 this.removeNode(node);
252
                 return true;
253
             let last_node = this.LastNode(node);
```

```
255
             let next_node = this.NextNode(node);
256
             this.splay(last_node);
257
             this.splay(next_node, last_node);
             this.removeNode(next_node.left);
258
259
             next_node.left = null;
             next_node.update();
260
261
             last_node.update();
262
             return true;
263
264 •
         has(value) { // O(logN)
265
             return this.count(value) > 0;
266
         count(value) { // O(logN)
267
             let x = this.findFirstOf(value);
268
269
             if (x == null) return 0;
270
             let rank_x = this.findRankOf(x);
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
         findKth(rank) { // (0-indexed) 0(logN)
279 •
280
             let x = this.findKthNode(rank);
281
             return x == null ? null : (x.val);
282
283 ▼
         higher(value) { // > upper_bound 0(logN)
284
             let node = this.findSuccessorOf(value);
285
             return node == null ? null : (node.val);
286
287 •
         lower(value) { // < O(logN)}
             let node = this.findPrecursorOf(value);
288
289
             return node == null ? null : (node.val);
290
291 🕶
         first() {
             return this.findKth(0);
292
293
294 •
         last() {
295
             return this.findKth(this.size() - 1);
296
         poll() {
297 •
298
             let res = this.first();
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() {
311
             return this.root == null;
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);
318
                  res.push(x.val);
319
                  dfs(x.right);
320
             };
321
             dfs(this.root);
322
             return res;
323
         }
324
325
326 v const continuousSubarrays = (a) ⇒ {
327
         let tree = new SplayTree(), n = a.length, l = 0, res = 0;
         for (let i = 0; i < n; i++) {
328 ▼
329
             tree.insert(a[i]);
             while (tree.last() - tree.first() > 2) tree.remove(a[l++]);
330
             res += i - l + 1;
```

```
7/2/23, 12:33 AM
                                                                         Continuous Subarrays - LeetCode Contest
   332
   333
              return res;
   334
         };
  ☐ Custom Testcase
                          Use Example Testcases
                                                                                                                                        Run
                                                                                                                                                   △ Submit
  Submission Result: Accepted (/submissions/detail/984264071/) ?
                                                                                     More Details > (/submissions/detail/984264071/)
  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)
```