

7022. Minimum Absolute Difference Between Elements With Constraint

My Submissions (/contest/weekly-contest-358/problems/minimum-absolute-difference-between-elements-with-constraint/submissions/)

Back to Contest (/contest/weekly-contest-358/)

You are given a **0-indexed** integer array `nums` and an integer `x`.

Find the **minimum absolute difference** between two elements in the array that are at least `x` indices apart.

In other words, find two indices `i` and `j` such that `abs(i - j) >= x` and `abs(nums[i] - nums[j])` is minimized.

Return an integer denoting the **minimum** absolute difference between two elements that are at least `x` indices apart.

User Accepted:	0
User Tried:	0
Total Accepted:	0
Total Submissions:	0
Difficulty:	Medium

Example 1:

Input: `nums = [4,3,2,4]`, `x = 2`
Output: `0`
Explanation: We can select `nums[0] = 4` and `nums[3] = 4`. They are at least 2 indices apart, and their absolute difference is the minimum, 0. It can be shown that 0 is the optimal answer.

Example 2:

Input: `nums = [5,3,2,10,15]`, `x = 1`
Output: `1`
Explanation: We can select `nums[1] = 3` and `nums[2] = 2`. They are at least 1 index apart, and their absolute difference is the minimum, 1. It can be shown that 1 is the optimal answer.

Example 3:

Input: `nums = [1,2,3,4]`, `x = 3`
Output: `3`
Explanation: We can select `nums[0] = 1` and `nums[3] = 4`. They are at least 3 indices apart, and their absolute difference is the minimum, 3. It can be shown that 3 is the optimal answer.

Constraints:

- `1 <= nums.length <= 105`
- `1 <= nums[i] <= 109`
- `0 <= x < nums.length`

JavaScript

db

↺

⚙

```
1 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() {
11    this.sz = (this.left != null ? this.left.sz : 0) + (this.right != null ? this.right.sz : 0) + 1;
12    this.sum = (this.left != null ? this.left.sum : 0) + (this.right != null ? this.right.sum : 0) + this.val;
13  }
14  isLeft() {
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  class SplayTree {
27      constructor() {
28          this.root = null;
29          this.cmp = (x, y) => x >= y ? 0 : 1;
30      }
31      zig(x) { // right rotation
32          let y = x.parent;
33          if (x.right != null) x.right.parent = y;
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      }
46      zag(x) { // left rotation
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()) {
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      }
61      zigzig(x) { // RR
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 node under guard, default splay to 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 {
86                  if (node.parent.isLeft()) {
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 }
100 }
101 if (guard == null) this.root = 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 }
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;
124     while (node != null) {
125         last_visited = node;
126         if (this.cmp(value, node.val)) {
127             node = node.left;
128         } else if (this.cmp(node.val, value)) {
129             node = node.right;
130         } else {
131             res = node;
132             node = node.left;
133         }
134     }
135     if (last_visited != null) this.splay(last_visited);
136     return res;
137 }
138 findLastOf(value) {
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;
144         } else if (this.cmp(node.val, value)) {
145             node = node.right;
146         } else {
147             res = node;
148             node = node.right;
149         }
150     }
151     if (last_visited != null) this.splay(last_visited);
152     return res;
153 }
154 findRankOf(node) {
155     this.splay(node);
156     return node.left == null ? 0 : node.left.sz;
157 }
158 findSuccessorOf(value) {
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     }
169     if (last_visited != null) this.splay(last_visited);
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;
188     let node = this.root;
189     while (node != null) {
190         let leftsize = node.left == null ? 0 : node.left.sz;
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 -----
210 insert(value) { // allow duplicates LST.set()
211     if (this.root == null) {
212         this.root = this.make(value);
213         return this.root;
214     }
215     let node = this.root;
216     while (node != null) {
217         if (this.cmp(value, node.val)) {
218             if (node.left == null) {
219                 node.left = this.make(value);
220                 node.left.parent = node;
221                 node = node.left;
222                 break;
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 }
238 remove(value) { // remove one node, not all LST.unset()
239     let node = this.find(value);
240     if (node == null) return false;
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);
246         return true;
247     }

```

```

248   if (node.right == null) {
249       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);
257   this.splay(next_node, last_node);
258   this.removeNode(next_node.left);
259   next_node.left = null;
260   next_node.update();
261   last_node.update();
262   return true;
263 }
264 has(value) { // LST.get()
265     return this.count(value) > 0;
266 }
267 count(value) {
268     let x = this.findFirstOf(value);
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
276     let x = this.findPrecursorOf(value);
277     return x == null ? 0 : this.findRankOf(x) + 1;
278 }
279 findKth(rank) { // (0-indexed)
280     let x = this.findKthNode(rank);
281     return x == null ? null : (x.val);
282 }
283 higher(value) { // > upper_bound() LST.next(value)
284     let node = this.findSuccessorOf(value);
285     return node == null ? null : (node.val);
286 }
287 lower(value) { // < LST.prev(value - 1)
288     let node = this.findPrecursorOf(value);
289     return node == null ? null : (node.val);
290 }
291 ceiling(value) { // >=
292     return this.has(value) ? value : this.higher(value);
293 }
294 floor(value) { // <=
295     return this.has(value) ? value : this.lower(value);
296 }
297 first() {
298     return this.findKth(0);
299 }
300 last() {
301     return this.findKth(this.size() - 1);
302 }
303 poll() {
304     let res = this.first();
305     this.remove(res);
306     return res;
307 }
308 pollLast() {
309     let res = this.last();
310     this.remove(res);
311     return res;
312 }
313 size() {
314     return this.root == null ? 0 : this.root.sz;
315 }
316 isEmpty() {
317     return this.root == null;
318 }
319 show() {
320     let res = [];
321     const dfs = (x) => {
322         if (x == null) return;
323         dfs(x.left);

```


```
324         res.push(x.val);
325         dfs(x.right);
326     };
327     dfs(this.root);
328     return res;
329 }
330 }
331
332 const minAbsoluteDifference = (a, x) => {
333     let n = a.length, tree = new SplayTree(), res = Number.MAX_SAFE_INTEGER;
334     for (let i = 0; i < n; i++) {
335         if (i - x >= 0) tree.insert(a[i - x]);
336         let low = tree.floor(a[i]), high = tree.ceiling(a[i]);
337         if (low != null) res = Math.min(res, a[i] - low);
338         if (high != null) res = Math.min(res, high - a[i]);
339     }
340     return res;
341 };
```

☐ Custom Testcase☒ Use Example Testcases Run SubmitSubmission Result: **Accepted** (/submissions/detail/1019875176/) ?

More Details > (/submissions/detail/1019875176/)

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 \(/region\)](#)