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/)

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

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 <= i < 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

```
class SplayNode {
    constructor(value) {
        this.parent = null;
        this.left = null;
        this.right = null;
        this.val = value;
        this.sum = value;
        this.sz = 1;
    }
    update() {
        this.sz = (this.left != null ? this.left.sz : 0) + (this.right != null ? this.right.sz : 0) + 1;
        this.sum = (this.left != null ? this.left.sum : 0) + (this.right != null ? this.right.sum : 0) + this.val;
    }
    isLeft() {
        return this.parent != null && this.parent.left == this;
    }
    isRight() {
        return this.parent != null && this.parent.right == this;
```

```
    }
    isRoot(guard = null) {
        return this.parent == guard;
    }
}

// MultiSet
class SplayTree {
    constructor() {
        this.root = null;
        this.cmp = (x, y) => x >= y ? 0 : 1;
    }
    zig(x) { // right rotation
        let y = x.parent;
        if (x.right != null) x.right.parent = y;
        y.left = x.right;
        x.right = y;
        if (y.isLeft()) {
            y.parent.left = x;
        } else if (y.isRight()) {
            y.parent.right = x;
        }
        x.parent = y.parent;
        y.parent = x;
        y.update();
        x.update();
    }
    zag(x) { // left rotation
        let y = x.parent;
        if (x.left != null) x.left.parent = y;
        y.right = x.left;
        x.left = y;
        if (y.isLeft()) {
            y.parent.left = x;
        } else if (y.isRight()) {
            y.parent.right = x;
        }
        x.parent = y.parent;
        y.parent = x;
        y.update();
        x.update();
    }
    zigzig(x) { // RR
        this.zig(x.parent);
        this.zig(x);
    }
    zigzag(x) { // RL
        this.zig(x);
        this.zag(x);
    }
    zagzag(x) { // LL
        this.zag(x.parent);
        this.zag(x);
    }
    zagzig(x) { // LR
        this.zag(x);
        this.zig(x);
    }
    splay(node, guard = null) { // splay a "node" just under a "guard", which is default to splay to the "root".
        while (!node.isRoot(guard)) {
            if (node.parent.isRoot(guard)) {
                if (node.isLeft()) {
                    this.zig(node);
                } else {
                    this.zag(node);
                }
            } else {
                if (node.parent.isLeft()) {
                    if (node.isLeft()) {
                        this.zigzig(node);
                    } else {
                        this.zagzig(node);
                    }
                } else {
                    if (node.isRight()) {
                        this.zagzag(node);
                    } else {
```

```
                        this.zigzag(node);
                    }
                }
            }
        }
        if (guard == null) this.root = node; // reset "root" to "node".
    }
    LastNode(x) {
        this.splay(x);
        let node = x.left;
        if (node == null) return null;
        while (node.right != null) node = node.right;
        this.splay(node);
        return node;
    }
    NextNode(x) {
        this.splay(x);
        let node = x.right;
        if (node == null) return null;
        while (node.left != null) node = node.left;
        this.splay(node);
        return node;
    }
    find(value) {
        return this.findFirstOf(value);
    }
    findFirstOf(value) {
        let node = this.root, res = null, last_visited = null;
        while (node != null) {
            last_visited = node;
            if (this.cmp(value, node.val)) {
                node = node.left;
            } else if (this.cmp(node.val, value)) {
                node = node.right;
            } else {
                res = node;
                node = node.left;
            }
        }
        if (last_visited != null) this.splay(last_visited);
        return res;
    }
    findLastOf(value) {
        let node = this.root, res = null, last_visited = null;
        while (node != null) {
            last_visited = node;
            if (this.cmp(value, node.val)) {
                node = node.left;
            } else if (this.cmp(node.val, value)) {
                node = node.right;
            } else {
                res = node;
                node = node.right;
            }
        }
        if (last_visited != null) this.splay(last_visited);
        return res;
    }
    findRankOf(node) {
        this.splay(node);
        return node.left == null ? 0 : node.left.sz;
    }
    findSuccessorOf(value) {
        let node = this.root, res = null, last_visited = null;
        while (node != null) {
            last_visited = node;
            if (this.cmp(value, node.val)) {
                res = node;
                node = node.left;
            } else {
                node = node.right;
            }
        }
        if (last_visited != null) this.splay(last_visited);
        return res;
    }
    findPrecursorOf(value) {
```

```
        let node = this.root, res = null, last_visited = null;
        while (node != null) {
            last_visited = node;
            if (this.cmp(node.val, value)) {
                res = node;
                node = node.right;
            } else {
                node = node.left;
            }
        }
        if (last_visited != null) this.splay(last_visited);
        return res;
    }
    findKthNode(rank) {
        if (rank < 0 || rank >= this.size()) return null;
        let node = this.root;
        while (node != null) {
            let leftsize = node.left == null ? 0 : node.left.sz;
            if (leftsize == rank) break;
            if (leftsize > rank) {
                node = node.left;
            } else {
                rank -= leftsize + 1;
                node = node.right;
            }
        }
        this.splay(node);
        return node;
    }
    make(value) {
        return new SplayNode(value);
    }
    removeNode(node) {
        node = null;
    }

    // ------------------------------- Public Usage -------------------------------------
    insert(value) { // allow duplicates, tree nodes allow same value O(logN)
        if (this.root == null) {
            this.root = this.make(value);
            return this.root;
        }
        let node = this.root;
        while (node != null) {
            if (this.cmp(value, node.val)) {
                if (node.left == null) {
                    node.left = this.make(value);
                    node.left.parent = node;
                    node = node.left;
                    break;
                }
                node = node.left;
            } else {
                if (node.right == null) {
                    node.right = this.make(value);
                    node.right.parent = node;
                    node = node.right;
                    break;
                }
                node = node.right;
            }
        }
        this.splay(node);
        return node;
    }
    remove(value) { // remove one node, not remove all O(logN)
        let node = this.find(value);
        if (node == null) return false;
        this.splay(node);
        if (node.left == null) {
            this.root = node.right;
            if (node.right != null) node.right.parent = null;
            this.removeNode(node);
            return true;
        }
        if (node.right == null) {
            this.root = node.left;
```

```
            if (node.left != null) node.left.parent = null;
            this.removeNode(node);
            return true;
        }
        let last_node = this.LastNode(node);
        let next_node = this.NextNode(node);
        this.splay(last_node);
        this.splay(next_node, last_node);
        this.removeNode(next_node.left);
        next_node.left = null;
        next_node.update();
        last_node.update();
        return true;
    }
    has(value) { // O(logN)
        return this.count(value) > 0;
    }
    count(value) { // O(logN)
        let x = this.findFirstOf(value);
        if (x == null) return 0;
        let rank_x = this.findRankOf(x);
        let y = this.findLastOf(value);
        let rank_y = this.findRankOf(y);
        return rank_y - rank_x + 1;
    }
    rankOf(value) { // The number of elements strictly less than value O(logN)
        let x = this.findPrecursorOf(value);
        return x == null ? 0 : this.findRankOf(x) + 1;
    }
    findKth(rank) { // (0-indexed) O(logN)
        let x = this.findKthNode(rank);
        return x == null ? null : (x.val);
    }
    higher(value) { // > upper_bound  O(logN)
        let node = this.findSuccessorOf(value);
        return node == null ? null : (node.val);
    }
    lower(value) { // <  O(logN)
        let node = this.findPrecursorOf(value);
        return node == null ? null : (node.val);
    }
    first() {
        return this.findKth(0);
    }
    last() {
        return this.findKth(this.size() - 1);
    }
    poll() {
        let res = this.first();
        this.remove(res);
        return res;
    }
    pollLast() {
        let res = this.last();
        this.remove(res);
        return res;
    }
    size() {
        return this.root == null ? 0 : this.root.sz;
    }
    isEmpty() {
        return this.root == null;
    }
    show() {  // Get sorted values in the splay tree O(n).
        let res = [];
        const dfs = (x) => {
            if (x == null) return;
            dfs(x.left);
            res.push(x.val);
            dfs(x.right);
        };
        dfs(this.root);
        return res;
    }
}

const sumImbalanceNumbers = (a) => {
```

```
    let n = a.length, res = 0;
    for (let i = 0; i < n; i++) {
        let tree = new SplayTree(), cnt = -1;
        for (let j = i; j < n; j++) {
            let x = a[j];
            if (!tree.has(x)) {
                tree.insert(x);
                cnt++;
                if (tree.has(x - 1)) cnt--;
                if (tree.has(x + 1)) cnt--;
            }
            res += cnt;
        }
    }
    return res;
};
```

Custom Testcase | Use Example Testcases

Run | Submit

Submission Result: Accepted (/submissions/detail/984304909/) | More Details > (/submissions/detail/984304909/)

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)