programmerall.com

[Leetcode] 315. Count of Smaller NumBers After Self calculates the number of smaller numbers

5-6 minutes

You are given an integer array *nums* and you have to return a new *counts* array. The *counts*array has the property where `counts[i]` is the number of smaller elements to the right of `nums[i]`.

Example:

Input: [5,2,6,1]
Output:

```
[2,1,1,0]
Explanation:
```

To the right of 5 there are 2 smaller elements (2 and 1).
To the right of 2 there is only 1 smaller element (1).
To the right of 6 there is 1 smaller element (1).
To the right of 1 there is 0 smaller element.

This topic gives an array that allows us to calculate all the numbers that are less than this number. The visual inspire cannot use brute force, oj certainly does not agree, so in order to improve the efficiency of the operation, first you can use two-point search

method, ideas It is a given group from the last beginning, inserted into a new array with two binarys, so that the new array is orderly, then the coordinates in the new array in the new array are all smaller numbers in the original number group. Number, see the code as follows:

Method 1:

```
// Binary Search
class Solution {
public:
    vector<int> countSmaller(vector<int>& nums) {
        vector<int> t, res(nums.size());
        for (int i = nums.size() - 1; i >= 0; --i) {
            int left = 0, right = t.size();
            while (left < right) {
                int mid = left + (right - left) / 2;
                if (t[mid] >= nums[i]) right = mid;
                else left = mid + 1;
            }
            res[i] = right;
            t.insert(t.begin() + right, nums[i]);
        }
        return res;
    }
};
```

The above use of the two-point search is an insertional practice, we can also use some of the STLs in C ++, such as seeking

distance distance, or the first function of the first unsained than the current number, here Using these two functions instead of the two-point search in the previous method, the core ideas of the two methods are the same, constructed ordered, to find newly added arrays in the ordered array corresponding location In the result, see the code as follows:

Solution 2:

```
// Insert Sort
class Solution {
public:
    vector<int> countSmaller(vector<int>& nums) {
        vector<int> t, res(nums.size());
        for (int i = nums.size() - 1; i >= 0; --i) {
            int d = distance(t.begin(), lower_bound(t.begin(), t.end(),
nums[i]));
            res[i] = d;
            t.insert(t.begin() + d, nums[i]);
        }
        return res;
    }
};
```

Take another way to use the two-point search tree, construct a two-point search tree, a slightly different place is to add a variable smaller to record all the numbers of all nodes than the current node value, each inserted one Node, it will determine the size of its and root nodes. If the new node value is smaller than the root

point value, it will be inserted into the left subtree, and the SMALLER of the root node is increased, and the recursive call is continued. INSERT of sub-node. If the node value is greater than the root point value, you need to recurrent inserting the INSERT of the right child and add the SMALLER of the root point, and add 1, see the code as follows:

Solution three:

```
// Binary Search Tree
class Solution {
public:
    struct Node {
        int val, smaller;
        Node *left, *right;
        Node(int v, int s) : val(v), smaller(s), left(NULL), right(NULL) {}
    };
    int insert(Node*& root, int val) {
        if (!root) return (root = new Node(val, 0)), 0;
        if (root->val > val) return root->smaller++, insert(root->left,
val);
        return insert(root->right, val) + root->smaller + (root->val < val
? 1 : 0);
    }
    vector<int> countSmaller(vector<int>& nums) {
        vector<int> res(nums.size());
        Node *root = NULL;
        for (int i = nums.size() - 1; i >= 0; --i) {
            res[i] = insert(root, nums[i]);
        }
```

```
        return res;
    }
};
```

GitHub sync address:

https://github.com/grandyang/leetcode/issues/315

Similar topic:

Count of Range Sum

Queue Reconstruction by Height

Reverse Pairs

Reference:

https://leetcode.com/problems/count-of-smaller-numbers-after-self/

https://leetcode.com/problems/count-of-smaller-numbers-after-self/discuss/76576/My-simple-AC-Java-Binary-Search-code

https://leetcode.com/problems/count-of-smaller-numbers-after-self/discuss/138154/The-C%2B%2B-merge-sort-template-for-pairs-'i'-'j'-problem

https://leetcode.com/problems/count-of-smaller-numbers-after-self/discuss/76611/Short-Java-Binary-Index-Tree-BEAT-97.33-With-Detailed-Explanation

https://leetcode.com/problems/count-of-smaller-numbers-after-self/discuss/76657/3-ways-(Segment-Tree-Binary-Indexed-Tree-Binary-Search-Tree)-clean-python-code

https://leetcode.com/problems/count-of-smaller-numbers-after-self/discuss/76607/C%2B%2B-O(nlogn)-Time-O(n)-Space-

MergeSort-Solution-with-Detail-Explanation

Leetcode All in One Total Solutions (Continuous Update ...)