581 Shortest Unsorted Continuous Subarray

2018年4月16日 11:43

Given an integer array, you need to find one **continuous subarray** that if you only sort this subarray in ascending order, then the whole array will be sorted in ascending order, too.

You need to find the **shortest** such subarray and output its length.

Example 1:

Input: [2, 6, 4, 8, 10, 9, 15]

Output: 5

Explanation: You need to sort [6, 4, 8, 10, 9] in ascending order to make the whole array sorted in ascending order.

Note:

1. Then length of the input array is in range [1, 10,000].
2. The input array may contain duplicates, so ascending order here means **<=**.

来自 <https://leetcode.com/problems/shortest-unsorted-continuous-subarray/description/>

给定一个整数数组，你需要寻找一个**连续的子数组**，如果对这个子数组进行升序排序，那么整个数组都会变为升序排序。

你找到的子数组应是**最短**的，请输出它的长度。

给定一个整数数组，你需要寻找一个**连续的子数组**，如果对这个子数组进行升序排序，那么整个数组都会变为升序排序。

你找到的子数组应是**最短**的，请输出它的长度。

Solution for Python3:

```python
class Solution1:
    def findUnsortedSubarray(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        srtNums = copy.deepcopy(nums)
        srtNums.sort()
        i, j = 0, len(srtNums) - 1
        while i < j and srtNums[i] == nums[i]:
            i += 1
        while i < j and srtNums[j] == nums[j]:
            j -= 1
        return 0 if i == j else j - i + 1

class Solution2:
    def findUnsortedSubarray(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        res = [a == b for a, b in zip(nums, sorted(nums))]
        return 0 if all(res) else len(nums) - res.index(False) - res[::-1].index(False)
```

Solution for C++:

```cpp
class Solution1 {
public:
    int findUnsortedSubarray(vector<int>& nums) {
        int l = nums.size(), r = 0;
        for (int i = 0; i < nums.size(); i++) {
            for (int j = i + 1; j < nums.size(); j++) {
                if (nums[i] > nums[j]) {
                    r = max(r, j);
                    l = min(l, i);
                }
            }
        }
        return r - l < 0 ? 0 : r - l + 1;
    }
};

class Solution2 {
public:
    int findUnsortedSubarray(vector<int>& nums) {
        stack<int> sta;
        int l = nums.size(), r = 0;
        for (int i = 0; i < nums.size(); i++) {
            while (!sta.empty() && nums[sta.top()] > nums[i]) {
                l = min(l, sta.top());
                sta.pop();
            }
            sta.push(i);
        }
        stack<int>().swap(sta);
        for (int i = nums.size() - 1; i >= 0; i--) {
            while(!sta.empty() && nums[sta.top()] < nums[i]) {
                r = max(r, sta.top());
                sta.pop();
            }
            sta.push(i);
        }
        return r - l > 0 ? r - l + 1 : 0;
    }
};

class Solution3 {
public:
    int findUnsortedSubarray(vector<int>& nums) {
        int minV = INT_MAX, maxV = INT_MIN;
        bool flag = false;
        for (int i = 1; i < nums.size(); i++) {
            if (nums[i] < nums[i - 1])
                flag = true;
            if (flag)
                minV = min(minV, nums[i]);
        }
        flag = false;
        for (int i = nums.size() - 2; i >=0 ; i--) {
            if (nums[i]> nums[i + 1])
                flag = true;
            if (flag)
                maxV = max(maxV, nums[i]);
        }
        int l, r;
        for (l = 0; l < nums.size(); l++) {
            if (minV < nums[l])
                break;
        }
        for (r = nums.size() - 1; r >= 0; r--) {
            if (maxV > nums[r])
                break;
        }
        return r - l < 0 ? 0 : r - l + 1;
    }
};

class Solution4 {
public:
    int findUnsortedSubarray(vector<int>& nums) {
        vector<int> srtNums = nums;
        sort(srtNums.begin(), srtNums.end());
        int i = 0, j = srtNums.size() - 1;
        while (i < j && srtNums[i] == nums[i])
            i++;
        while (i < j && srtNums[j] == nums[j])
            j--;
        if (i == j)
            return 0;
        return j - i + 1;
    }
};
```