Non-decreasing Array

Given an array with `n` integers, your task is to check if it could become non-decreasing by modifying **at most** `1` element.

We define an array is non-decreasing if `array[i] holds for every i (1`

`Example 1:`

```
Input: [4,2,3]
Output: True
Explanation: You could modify the first 4 to 1 to get a non-decreasing array.
```

Example 2:

```
Input: [4,2,1]
Output: False
Explanation: You can't get a non-decreasing array by modify at most one element.
```

Note: The `n` belongs to [1, 10,000].

Solution 1

This problem is like a greedy problem. When you find `nums[i-1] > nums[i]` for some `i`, you will prefer to change `nums[i-1]`'s value, since a larger `nums[i]` will give you more risks that you get inversion errors after position `i`. But, if you also find `nums[i-2] > nums[i]`, then you have to change `nums[i]`'s value instead, or else you need to change both of `nums[i-2]`'s and `nums[i-1]`'s values.

Java version:

```java
 public boolean checkPossibility(int[] nums) {
        int cnt = 0;
//the number of changes
        for(int i = 1; i < nums.length && cnt<=1 ; i++){
            if(nums[i-1] > nums[i]){
                cnt++;
                if(i-2<0 || nums[i-2] <= nums[i])nums[i-1] = nums[i];
//modify nums[i-1] of a priority
                else nums[i] = nums[i-1];
//have to modify nums[i]
            }
        }
        return cnt<=1;
    }
```

C++ version:

```cpp
bool checkPossibility(vector<int>& nums) {
        int cnt = 0;
//the number of changes
        for(int i = 1; i < nums.size() && cnt<=1 ; i++){
            if(nums[i-1] > nums[i]){
                cnt++;
                if(i-2<0 || nums[i-2] <= nums[i])nums[i-1] = nums[i];
//modify nums[i-1] of a priority
                else nums[i] = nums[i-1];
//have to modify nums[i]
            }
        }
        return cnt<=1;
    }
```

written by Hao Cai original link here

Solution 2

First, find a pair where the order is wrong. Then there are two possibilities, either the first in the pair can be modified or the second can be modified to create a valid sequence. We simply modify both of them and check for validity of the modified arrays.

I find this approach the easiest to reason about and understand.

By Yang Shun

```
class Solution(object):
    def checkPossibility(self, nums):
        """
        :type nums: List[int]
        :rtype: bool
        """
        one, two = nums[:], nums[:]
        for i in range(len(nums) - 1):
            if nums[i] > nums[i + 1]:
                one[i] = nums[i + 1]
                two[i + 1] = nums[i]
                break
        def valid(arr):
            for i in range(len(arr) - 1):
                if arr[i] > arr[i + 1]:
                    return False
            return True
        return valid(one) or valid(two)
```

written by yangshun original link here

Solution 3

The strategy is to `lower a[i-1]` to match `a[i]` if possible - (`a[i-2]` not exist or no smaller than `a[i]`);
otherwise `rise a[i]` to match `a[i-1]` .

```
2 Examples:
 0  ...  i ...
 1 1 2 4[2]5 6  - in this case we can just raise a[i] to 4;
         4
 1 1 2[4]2 3 4  - in this case lower a[i-1] is better;
       2
```

Java - Modifying Input

```
class Solution {
    public boolean checkPossibility(int[] a) {
        int modified = 0;
        for (int i = 1; i < a.length; i++) {
            if (a[i] < a[i - 1]) {
                if (modified++ > 0) return false;
                if (i - 2 < 0 || a[i - 2] <= a[i]) a[i - 1] = a[i]; // lower a[i -
1]
                else a[i] = a[i - 1]; // rise a[i]
            }
        }
        return true;
    }
}
```

We can also do it without modifying the input by using a variable `prev` to hold the `a[i-1]` ; if we have to `lower a[i]` to match `a[i-1]` instead of `raising a[i-1]` , simply skip updating `prev` ;

Java - Without Modifying Input

```
class Solution {
    public boolean checkPossibility(int[] a) {
        int modified = 0;
        for (int i = 1, prev = a[0]; i < a.length; i++) {
            if (a[i] < prev) {
                if (modified++ > 0) return false;
                if (i - 2 >= 0 && a[i - 2] > a[i]) continue;
            }
            prev = a[i];
        }
        return true;
    }
}
```

Or

```
class Solution {
    public boolean checkPossibility(int[] a) {
        int modified = 0;
        for (int i = 1, prev = a[0]; i < a.length; i++) {
            if (a[i] < prev && modified++ > 0) return false;
            if (a[i] < prev && i - 2 >= 0 && a[i - 2] > a[i]) continue;
            prev = a[i];
        }
        return true;
    }
}
```

C++ - Modifying Input

```
class Solution {
public:
    bool checkPossibility(vector<int>& a) {
        bool modified = false;
        for (int i = 1; i < a.size(); i++) {
            if (a[i] < a[i - 1]) {
                if (modified++) return false;
                (i - 2 < 0 || a[i - 2] <= a[i]) ? a[i - 1] = a[i] : a[i] = a[i - 1]
;
            }
        }
        return true;
    }
};
```

C++ - Without Modifying Input

```
class Solution {
public:
    bool checkPossibility(vector<int>& a) {
        bool modified = false;
        for (int i = 1, prev = a[0]; i < a.size(); i++) {
            if (a[i] < prev && modified++) return false;
            if (a[i] < prev && i - 2 >= 0 && a[i - 2] > a[i]) continue;
            prev = a[i];
        }
        return true;
    }
};
```

written by alexander original link here