SQP6 Next Greatest Element I

Friday, November 25, 2022 10:08 AM

The **next greater element** of some element x in an array is the **first greater** element that is **to the right** of x in the same array.
You are given two **distinct 0-indexed** integer arrays nums1 and nums2, where nums1 is a subset of nums2.

For each 0 <= i < nums1.length, find the index j such that nums1[i] == nums2[j] and determine the **next greater element** of nums2[j] in nums2. If there is no next greater element, then the answer for this query is -1.

Return *an array* ans *of length* nums1.length *such that* ans[i] *is the* ***next greater element*** *as described above.*

Example 1:
Input: nums1 = [4,1,2], nums2 = [1,3,4,2]
Output: [-1,3,-1]

↳ What is the 1st variant type of problem based on Stack and queue

Explanation: The next greater element for each value of nums1 is as follows:
- 4 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the answer is -1.
- 1 is underlined in nums2 = [1,3,4,2]. The next greater element is 3.
- 2 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the answer is -1.

In this problem we find the every element of Greater element

Example

n = 11
arr = [3 10 4 2 1 2 6 1 7 2 9]

→ Now Our task is finding the Next Greatest element of every element from the array

arr = [3 10 4 2 1 2 6 1 7 2 9] Next Greatest element According to this element is

Next Greatest Element = [10 -1 6 6 2 6 7 7 9 9 10]

↳ That is known as Next Greatest Element of every element if exist

→ This is the Circular kind of Array problem or Stack problem

Thought Process

→ Now first we try to the solving using the simplest way or easy Approach then try do the Optimial Approch

1st Variant

The first Variant of the problem we Not Considered is a Circular kind of problem

→ So every Iteration (i) We will Check the (i+1) to len(arr) first Greater element for arr[i] Otherwise put -1 →

Example

arr = [3 10 4 2 1 2 6 1 7 2 9]
(i)
ans = [10 -1 6 6 2 6 7 7 9 9 10]

↳ this is the Circular kind of problem

→ In this approach we only put the Next Greatest

→ this is the circular way of problem

→ In this approach we only put the Next Greatest element if (i) is the last element of the array so put -1

ans = [10 -1 6 6 2 6 7 7 9 9 -1] ✓

Thought Process

→ According to the brute force Approach we required (2) loop

for (i=0 to i=N)
  for (i+1 to i=N)

→ In this case we used the (N^2) time Complexity that is Not a good Approch for this problem

TimeComplicity = $O(N^2)$

Optimized Approch

→ $O(N^2)$ that is not Good Approach so Interviewer asked to Optimized that Approch

→ In this Case required the Stack data Structure.
→ We required the Stack data Structure

Stack = 

let is Understand the Algorithm

arr = [4,12,5,3,1,2,5,3,1,2,4,6]

→ Now we Start the Over Iteration from the end of the array
→ and apply same condition on the stack

→ Now we Stack Iterate from ← left hand side of the array

→ if Stack isEmpty()
  → So simply assign that Index of (-1) and put that element into the Stack

→ Every Iteration of (i) we check if any element present into the Stack that is Smaller than (i) so Simply removed all the element that arr < arr[i]

→ Otherwise take the top element from the Stack and put into that Index and put (i) into the Stack

→ Now that approach follows again and again till the over i == 0 or end of the array

Dry Run

of the array

Dry Run

arr = | 4 | 12 | 5 | 3 | 1 | 2 | 5 | 3 | 1 | 2 | 4 | 6 |
i

→ Now initially we start the end of the array and our Stack is empty

Stack =

→ required one more array of the size of the array from start the last answer

arr = | 4 | 12 | 5 | 3 | 1 | 2 | 5 | 3 | 1 | 2 | 4 | 6 |
↑ i

ans = | | | | | | | | | | | 6 | -1 |

i = 6 So check one if the stack if empty() ✓

→ put the -1 that means not any element present into the Stack so put -1 into the answer and put arr[i] into the Stack

Stack =
6

i = 4 So check one if the stack is Empty() X

→ Check if any element present into the stack that is smaller than arr[i] X

→ that mean pop the element from the Stack (take the top element from the Stack) and assigine into the index of answer and put the arr[i] into the stack

Stack =
4
6

i = 5 again check if stack if empty() X

→ check if any element present into the Stack that is smaller than arr[i] ✓

→ So basically pop that all element from the stack that is smaller than arr[i]

→ after that take the top element from the Stack and put into the index of i into the answer and put arr[i] into the stack

Stack =
5

Stack = 5
6

→ Now that is the approach that we follows again and again till the end of the array

TimeComplexity = (2N + 2N) ≈ O(N)
↳ for loop
↳ while loop

SpaceComplexity = O(N)
↳ Stack + answer we used extra Spaced

Important Point
Why we removed smaller element from the stack = ???

↳ because Commonly we used that our observation
↳ removed smaller element because if we have
0 7 1 2 3
↑
↳ that is my Current Greatest element and i traversal into the → direction so (7) came first

(7) that is the my Greatest element so we not only used of (123) because we not reach there that why we removed all the smaller element from the stack

Implementation

```
# lc: https://leetcode.com/problems/next-greater-element-ii/
# gfg: https://practice.geeksforgeeks.org/problems/next-larger-element-1587115620/1

# todo varient I typle not circular

class Solution:
    def nextLargerElement(self, nums, n):

        ans = [0] * n
        stack = []

        for i in range(n - 1, -1, -1):

            while len(stack) != 0 and stack[-1] <= nums[i]:
                stack.pop()

            if i < n:
                if len(stack) != 0:
                    ans[i] = stack[-1]
                else:
                    ans[i] = -1

            stack.append(nums[i])

        return ans
```

Now come to the over Original problem that is Variant 2 (Circular array or Stack)

→ We used the same previously concept here but different manner

Circular Array

arr = [2, 10, 12, 1, 11]

→ basically in this problem add one more time of same array at the end of the original array

arr = [2, 10, 12, 1, 11, 2, 10, 12, 1, 11]

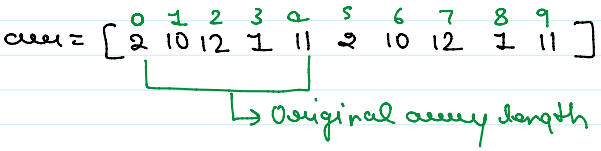
↳ same array copy at the end of the array

→ after adding the end of same array used same previous approach but different manner

→ in this approach we iterate at the forward direction ⟶ of the array

→ previously we learn iterate of the end of the array backward direction ⟵ of the array

→ and iterate only original length of the array (next duplicate) combined array



ans = [10 12 -1 11 12]

↳ that is the my answer what we want it

ans = [10 12 -1 11 12]
↳ that is the my answer
what we want it

→ Now our duplicate array that we copied at the end of the array that is our imaginary array so

(N=5) → original length

↳ So what can I do is every time I take the (i%n) for the perfect index

Example

i=5
i%n = 5%5 = (0)
that means our value of i is arr[0]

i=7
i%n = 7%5 = (2)
that means our value of i is arr[2]

i=9
i%n = 9%5 = (4)
that means our value of i is arr[4] ☺

→ that Means we can say we iterate (2xN) times

→ and working on the concept of the (i%N) we not required to do the double the array

Time Complexity = O(2N+2N) ≈ O(N)
↳ here using our two loop that run (2N) times
↳ two while loop that while loop not run always at 2 times

We not required to duplicate the array because we work on the Principle —
↳ Imaginary array Concept
↳ (i%N) ☺

→ while loop run same time 1,2,3,4 ----(2N-1) times

→ Our all unite tested Number of times iterate that is total Number of element into the array

→ During the code take (count) under the while loop and print every time during the while loop execution then we better understand how many times while loop run

Space Complexity = O(N)
↳ because we used extra Space to store the answer

9

#Implementation Python

#Implementation Java

class Solution{

Implementation Python

```
# lc: https://leetcode.com/problems/next-greater-element-ii/
# gfg:https://practice.geeksforgeeks.org/problems/next-larger-element-1587115620/1

# todo varient II typle its circular
class Solution:
    def nextGreaterElements(self, nums):

        n = len(nums)
        ans = [0] * n
        stack = []
        for i in range((2 * n) - 1, -1, -1):

            while len(stack) != 0 and stack[-1] <= nums[i % n]:
                stack.pop()

            if i < n:
                if len(stack) != 0:
                    ans[i] = stack[-1]
                else:
                    ans[i] = -1

            stack.append(nums[i % n])

        return ans
```

Implementation Java

```
class Solution{
    public int[] nextGreaterElements(int[] nums) {

        int n = nums.length;
        int[] ans = new int[n];
        Stack < Integer > st = new Stack < > ();

        for(int i = (2*n)-1;i >= 0;i--){
            while(st.isEmpty() == false && st.peek() <= nums[i % n]){
                st.pop();
            }
            if(i < n){
                if(st.isEmpty() == false){
                    ans[i] = st.peek();
                }
                else{
                    ans[i] = -1;
                }
            }

            st.push(nums[i%n]);
        }

        return ans;

    }
}
```