Problem 496: Next Greater Element I

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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]

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.

Example 2:

Input:

nums1 = [2,4], nums2 = [1,2,3,4]

Output:

[3,-1]

Explanation:

The next greater element for each value of nums1 is as follows: - 2 is underlined in nums2 = [1,

2

,3,4]. The next greater element is 3. - 4 is underlined in nums2 = [1,2,3,

4

]. There is no next greater element, so the answer is -1.

Constraints:

1 <= nums1.length <= nums2.length <= 1000

0 <= nums1[i], nums2[i] <= 10

4

All integers in

nums1

and

nums2

are

unique

.

All the integers of

nums1

also appear in

nums2

.

Follow up:

Could you find an

O(nums1.length + nums2.length)

solution?

Code Snippets

C++:

```
class Solution {
public:
vector<int> nextGreaterElement(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java:

```
class Solution {
public int[] nextGreaterElement(int[] nums1, int[] nums2) {

}
}
```

Python3:

```
class Solution:
def nextGreaterElement(self, nums1: List[int], nums2: List[int]) ->
List[int]:
```

Python:

```
class Solution(object):
def nextGreaterElement(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} nums1
* @param {number[]} nums2
* @return {number[]}
*/
var nextGreaterElement = function(nums1, nums2) {

};
```

TypeScript:

```
function nextGreaterElement(nums1: number[], nums2: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] NextGreaterElement(int[] nums1, int[] nums2) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* nextGreaterElement(int* nums1, int nums1Size, int* nums2, int nums2Size,
int* returnSize) {

}
```

Go:

```
func nextGreaterElement(nums1 []int, nums2 []int) []int {

}
```

Kotlin:

```
class Solution {
fun nextGreaterElement(nums1: IntArray, nums2: IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func nextGreaterElement(_ nums1: [Int], _ nums2: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn next_greater_element(nums1: Vec<i32>, nums2: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Integer[]}
def next_greater_element(nums1, nums2)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @return Integer[]
*/
function nextGreaterElement($nums1, $nums2) {

}
}
```

Dart:

```
class Solution {
List<int> nextGreaterElement(List<int> nums1, List<int> nums2) {

}
}
```

Scala:

```
object Solution {
def nextGreaterElement(nums1: Array[Int], nums2: Array[Int]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec next_greater_element(nums1 :: [integer], nums2 :: [integer]) ::
[integer]
def next_greater_element(nums1, nums2) do

end
end
```

Erlang:

```
-spec next_greater_element(Nums1 :: [integer()], Nums2 :: [integer()]) ->
[integer()].
next_greater_element(Nums1, Nums2) ->
.
```

Racket:

```
(define/contract (next-greater-element nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Next Greater Element I
* Difficulty: Easy
* Tags: array, hash, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<int> nextGreaterElement(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Next Greater Element I
* Difficulty: Easy
* Tags: array, hash, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int[] nextGreaterElement(int[] nums1, int[] nums2) {

}
}
```

Python3 Solution:

```
"""
Problem: Next Greater Element I
Difficulty: Easy
Tags: array, hash, stack
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def nextGreaterElement(self, nums1: List[int], nums2: List[int]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def nextGreaterElement(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Next Greater Element I
* Difficulty: Easy
* Tags: array, hash, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} nums1
* @param {number[]} nums2
* @return {number[]}
*/
var nextGreaterElement = function(nums1, nums2) {
};
```

TypeScript Solution:

```
/**
* Problem: Next Greater Element I
* Difficulty: Easy
* Tags: array, hash, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function nextGreaterElement(nums1: number[], nums2: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Next Greater Element I
* Difficulty: Easy
* Tags: array, hash, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int[] NextGreaterElement(int[] nums1, int[] nums2) {

}
}
```

C Solution:

```
/*
* Problem: Next Greater Element I
* Difficulty: Easy
* Tags: array, hash, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* nextGreaterElement(int* nums1, int nums1Size, int* nums2, int nums2Size,
int* returnSize) {

}
```

Go Solution:

```
// Problem: Next Greater Element I
// Difficulty: Easy
// Tags: array, hash, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func nextGreaterElement(nums1 []int, nums2 []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun nextGreaterElement(nums1: IntArray, nums2: IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func nextGreaterElement(_ nums1: [Int], _ nums2: [Int]) -> [Int] {
}
}
```

Rust Solution:

```
// Problem: Next Greater Element I
// Difficulty: Easy
// Tags: array, hash, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn next_greater_element(nums1: Vec<i32>, nums2: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Integer[]}
def next_greater_element(nums1, nums2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @return Integer[]
*/
function nextGreaterElement($nums1, $nums2) {

}
}
```

Dart Solution:

```
class Solution {
List<int> nextGreaterElement(List<int> nums1, List<int> nums2) {

}
}
```

Scala Solution:

```
object Solution {
def nextGreaterElement(nums1: Array[Int], nums2: Array[Int]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec next_greater_element(nums1 :: [integer], nums2 :: [integer]) ::
[integer]
def next_greater_element(nums1, nums2) do

end
end
```

Erlang Solution:

```
-spec next_greater_element(Nums1 :: [integer()], Nums2 :: [integer()]) ->
[integer()].
next_greater_element(Nums1, Nums2) ->
.
```

Racket Solution:

```
(define/contract (next-greater-element nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```