Problem 2899: Last Visited Integers

Problem Information

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

Problem Description

Given an integer array

nums

where

nums[i]

is either a positive integer or

-1

. We need to find for each

-1

the respective positive integer, which we call the last visited integer.

To achieve this goal, let's define two empty arrays:

seen

and

ans

.

Start iterating from the beginning of the array

nums

.

If a positive integer is encountered, prepend it to the

front

of

seen

.

If

-1

is encountered, let

k

be the number of

consecutive

-1

s seen so far (including the current

-1

),

If

k

is less than or equal to the length of

seen

, append the

k

-th element of

seen

to

ans

.

If

k

is strictly greater than the length of

seen

, append

-1

to

ans

.

Return the array

ans

.

Example 1:

Input:

nums = [1,2,-1,-1,-1]

Output:

[2,1,-1]

Explanation:

Start with

seen = []

and

ans = []

.

Process

nums[0]

: The first element in nums is

1

. We prepend it to the front of

seen

. Now,

seen == [1]

.

Process

nums[1]

: The next element is

2

. We prepend it to the front of

seen

. Now,

seen == [2, 1]

.

Process

nums[2]

: The next element is

-1

. This is the first occurrence of

-1

, so

k == 1

. We look for the first element in seen. We append

2

to

ans

. Now,

ans == [2]

.

Process

nums[3]

: Another

-1

. This is the second consecutive

-1

, so

k == 2

. The second element in

seen

is

1

, so we append

1

to

ans

. Now,

ans == [2, 1]

.

Process

nums[4]

: Another

-1

, the third in a row, making

k = 3

. However,

seen

only has two elements (

[2, 1]

). Since

k

is greater than the number of elements in

seen

, we append

-1

to

ans

. Finally,

ans == [2, 1, -1]

.

Example 2:

Input:

nums = [1,-1,2,-1,-1]

Output:

[1,2,1]

Explanation:

Start with

seen = []

and

ans = []

.

Process

nums[0]

: The first element in nums is

1

. We prepend it to the front of

seen

. Now,

seen == [1]

.

Process

nums[1]

: The next element is

-1

. This is the first occurrence of

-1

, so

k == 1

. We look for the first element in

seen

, which is

1

. Append

1

to

ans

. Now,

ans == [1]

.

Process

nums[2]

: The next element is

2

. Prepend this to the front of

seen

. Now,

seen == [2, 1]

.

Process

nums[3]

: The next element is

-1

. This

-1

is not consecutive to the first

-1

since

2

was in between. Thus,

k

resets to

1

. The first element in

seen

is

2

, so append

2

to

ans

. Now,

ans == [1, 2]

.

Process

nums[4]

: Another

-1

. This is consecutive to the previous

-1

, so

k == 2

. The second element in

seen

is

1

, append

1

to

ans

. Finally,

ans == [1, 2, 1]

.

Constraints:

1 <= nums.length <= 100

nums[i] == -1

or

1 <= nums[i] <= 100

Code Snippets

C++:

```
class Solution {
public:
vector<int> lastVisitedIntegers(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public List<Integer> lastVisitedIntegers(int[] nums) {

}
}
```

Python3:

```
class Solution:
def lastVisitedIntegers(self, nums: List[int]) -> List[int]:
```

Python:

```
class Solution(object):
def lastVisitedIntegers(self, nums):
"""
:type nums: List[int]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @return {number[]}
*/
var lastVisitedIntegers = function(nums) {

};
```

TypeScript:

```
function lastVisitedIntegers(nums: number[]): number[] {

};
```

C#:

```
public class Solution {
public IList<int> LastVisitedIntegers(int[] nums) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* lastVisitedIntegers(int* nums, int numsSize, int* returnSize) {

}
```

Go:

```
func lastVisitedIntegers(nums []int) []int {

}
```

Kotlin:

```
class Solution {
fun lastVisitedIntegers(nums: IntArray): List<Int> {

}
}
```

Swift:

```
class Solution {
func lastVisitedIntegers(_ nums: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn last_visited_integers(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer[]}
def last_visited_integers(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer[]
*/
function lastVisitedIntegers($nums) {

}
}
```

Dart:

```
class Solution {
List<int> lastVisitedIntegers(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def lastVisitedIntegers(nums: Array[Int]): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec last_visited_integers(nums :: [integer]) :: [integer]
def last_visited_integers(nums) do

end
end
```

Erlang:

```
-spec last_visited_integers(Nums :: [integer()]) -> [integer()].
last_visited_integers(Nums) ->
.
```

Racket:

```
(define/contract (last-visited-integers nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Last Visited Integers
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<int> lastVisitedIntegers(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Last Visited Integers
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public List<Integer> lastVisitedIntegers(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Last Visited Integers
Difficulty: Easy
Tags: array

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""


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

Python Solution:

```
class Solution(object):
def lastVisitedIntegers(self, nums):
"""
:type nums: List[int]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Last Visited Integers
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
```

```
* @param {number[]} nums
* @return {number[]}
*/
var lastVisitedIntegers = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Last Visited Integers
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function lastVisitedIntegers(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Last Visited Integers
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public IList<int> LastVisitedIntegers(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Last Visited Integers
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* lastVisitedIntegers(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Last Visited Integers
// Difficulty: Easy
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func lastVisitedIntegers(nums []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun lastVisitedIntegers(nums: IntArray): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func lastVisitedIntegers(_ nums: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Last Visited Integers
// Difficulty: Easy
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn last_visited_integers(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer[]}
def last_visited_integers(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer[]
*/
function lastVisitedIntegers($nums) {

}
}
```

Dart Solution:

```
class Solution {
List<int> lastVisitedIntegers(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def lastVisitedIntegers(nums: Array[Int]): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec last_visited_integers(nums :: [integer]) :: [integer]
def last_visited_integers(nums) do

end
end
```

Erlang Solution:

```
-spec last_visited_integers(Nums :: [integer()]) -> [integer()].
last_visited_integers(Nums) ->
.
```

Racket Solution:

```
(define/contract (last-visited-integers nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```