Problem 2899: Last Visited Integers

Problem Information

Difficulty: Easy
Acceptance Rate: 61.77%
Paid Only: No
Tags: Array, Simulation

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 = []`.

1. Process `nums[0]`: The first element in nums is `1`. We prepend it to the front of `seen`. Now, `seen == [1]`. 2. Process `nums[1]`: The next element is `2`. We prepend it to the front

of `seen`. Now, `seen == [2, 1]`. 3. 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]`. 4. 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]`. 5. 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 = []`.

1. Process `nums[0]`: The first element in nums is `1`. We prepend it to the front of `seen`. Now, `seen == [1]`. 2. 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]`. 3. Process `nums[2]`: The next element is `2`. Prepend this to the front of `seen`. Now, `seen == [2, 1]`. 4. 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]`. 5. 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]:
```