485. Max Consecutive Ones

Easy Array

Leetcode Link

Problem Description

You are given an array `nums` that consists only of 0s and 1s. Your task is to find the length of the longest subsequence of consecutive 1s in this array. This means you want to find the longest sequence where 1s appear one after another with no 0s between them.

For example: If the input is `nums = [1, 1, 0, 1, 1, 1]`, the consecutive 1s are `[1, 1]`, `[1, 1, 1]`. Among these, the longest sequence of 1s is `[1, 1, 1]`, which has a length of 3. Thus, the output should be `3`.

Intuition

To solve this problem, we perform a single pass through the array, using a counter to keep track of the number of consecutive 1s found at any point. As we iterate over the array, we follow this process:

1. When we encounter a `1`, we increase our counter by 1 because we have found another consecutive 1.
2. When we encounter a `0`, it means the current sequence of consecutive 1s has ended. At this point, we compare our current counter with the maximum length found so far (stored in a separate variable), updating the maximum length if necessary. After that, we reset our counter to zero as we are starting a new sequence from scratch.
3. After the iteration, we compare and update the maximum length one final time, as the longest sequence of 1s might end at the last element of the array, and no zero would be encountered to trigger the update of the maximum length.
4. Finally, we return the maximum length of consecutive 1s found.

This approach works because we are interested in the longest sequence of consecutive 1s, and we're keeping track of the length of the current sequence and the maximum found so far.

Solution Approach

The solution uses a straightforward linear scanning algorithm. It's a single pass through the given array with a time complexity of `O(n)`—`n` being the number of elements in the input array. No additional data structures are required for this solution, as we can solve the problem using two integer variables: one for the current count of consecutive 1s (`cnt`) and another for storing the maximum found so far (`ans`).

Here are the details of the solution approach step by step:

1. Initialize two variables `cnt` and `ans` to `0`. The `cnt` variable is used to keep track of the current count of consecutive 1s as we iterate through the array, while `ans` is used to store the maximum count of consecutive 1s encountered so far.
2. Iterate through each element `v` in the input array `nums`.
3. For each element `v`:
   - If `v` is `1`, this means we have encountered a consecutive 1, so we increment `cnt` by `1`.
   - If `v` is not `1` (which means `v` is `0`), we've reached the end of the current sequence of consecutive 1s, and we need to update `ans` with the maximum count so far: `ans = max(ans, cnt)`. Then, we reset `cnt` to `0` as we want to start counting a new sequence of consecutive 1s.
4. After the loop, we perform one last update to `ans`. This step is crucial as the longest sequence of consecutive 1s might end with the last element of the array, so there wouldn't be a `0` to trigger the update inside the loop. Therefore, we need to ensure `ans` also takes into account the count of the last sequence of 1s: `ans = max(ans, cnt)`.
5. Return `ans` as the final result, which is the maximum number of consecutive 1s found in the array.

The algorithm leverages the simplicity of the problem statement by maintaining a running count and updating the maximum as needed, which avoids the use of additional space and ensures an optimal time complexity.

Example Walkthrough

Let's consider a small example to understand the solution approach. Suppose the input array is `nums = [0, 1, 1, 0, 1, 0, 1, 1, 1, 0]`. We want to find the length of the longest subsequence of consecutive 1's.

Here's how the algorithm works step by step for this example:

1. Initialize `cnt` and `ans` to `0`. At this stage, `cnt = 0` and `ans = 0`.
2. Start iterating through each element `v` in `nums`:
   - `v = 0` (1st element): `cnt` remains `0` as we haven't encountered a `1` yet.
   - `v = 1` (2nd element): Increment `cnt` to `1`. No need to update `ans` yet (`ans = 0`).
   - `v = 1` (3rd element): Increment `cnt` to `2`. No need to update `ans` yet (`ans = 0`).
   - `v = 0` (4th element): Sequence of `1`'s ended. Update `ans` to `max(0, 2)`, which is `2`. Reset `cnt` to `0`.
3. Continue the process for remaining elements:
   - `v = 1` (5th element): Increment `cnt` to `1`.
   - `v = 0` (6th element): Sequence of `1`'s ended. Update `ans` to `max(2, 1)`, which remains `2`. Reset `cnt` to `0`.
   - `v = 1` (7th element): Increment `cnt` to `1`.
   - `v = 1` (8th element): Increment `cnt` to `2`.
   - `v = 1` (9th element): Increment `cnt` to `3`.
   - `v = 0` (10th element): Sequence of `1`'s ended. Update `ans` to `max(2, 3)`, which is `3`. Reset `cnt` to `0`.
4. Iteration is complete. Last update `ans` to `max(ans, cnt)` one final time, in case the longest sequence ended with the last element. But in this case, the longest sequence (`ans = 3`) had already been updated and `cnt` has been reset to `0`. Therefore, `ans` remains `3`.
5. Return `ans`, which is `3`, the length of the longest subsequence of consecutive 1s found in the array.

By updating `ans` each time we reach the end of a consecutive sequence of `1`'s and after the final element, the algorithm effectively finds the longest subsequence without the need for additional memory. This example illustrates the algorithm correctly tracking and updating the length of consecutive `1`'s sequences.

Python Solution

```python
class Solution:
    def findMaxConsecutiveOnes(self, nums: List[int]) -> int:
        # Initialize counters for the current sequence of ones (current_count) and
        # the maximum sequence found (max_count).
        current_count = max_count = 0

        # Iterate through each number in the input list.
        for value in nums:
            # If the current number is 1, increment the current sequence counter.
            if value == 1:
                current_count += 1
            else:
                # If the current number is not 1, update the maximum sequence counter
                # if the current sequence is the longest seen so far.
                max_count = max(max_count, current_count)
                # Reset current sequence counter to 0 as the sequence of ones has been broken.
                current_count = 0

        # After iterating through the list, check once more if the last sequence of ones
        # is the longest as it could end with the list.
        return max(max_count, current_count)

# Note: In this implementation, 'nums' is expected to be a list of integers where each integer is 0 or 1.
```

Java Solution

```java
class Solution {
    public int findMaxConsecutiveOnes(int[] nums) {
        // Initialize count of consecutive ones
        int currentCount = 0;
        // Initialize the maximum count of consecutive ones
        int maxCount = 0;

        // Iterate over each element in the array
        for (int value : nums) {
            if (value == 1) {
                // If the current element is 1, increment the current count
                currentCount++;
            } else {
                // If the current element is not 1, update the maxCount if the current count is greater than maxCount
                maxCount = Math.max(maxCount, currentCount);
                // Reset the current count to zero
                currentCount = 0;
            }
        }

        // In case the array ends with a sequence of ones, make sure to update the maxCount
        maxCount = Math.max(currentCount, maxCount);

        // Return the maximum count of consecutive ones found in the array
        return maxCount;
    }
}
```

C++ Solution

```cpp
#include <vector>
#include <algorithm> // Include algorithm library to use max function

class Solution {
public:
    // Function to find the maximum number of consecutive ones in the vector 'nums'
    int findMaxConsecutiveOnes(vector<int>& nums) {
        int currentCount = 0;    // Tracks the current sequence length of consecutive ones
        int maxCount = 0;        // Stores the maximum sequence length found

        // Loop over each element in the input vector
        for (int value : nums) {
            if (value == 1) {
                // If the current element is 1, we increment the current sequence length
                ++currentCount;
            } else {
                // If the current element is 0, find the maximum of 'maxCount' and 'currentCount',
                // then reset 'currentCount' for the next sequence of ones
                maxCount = std::max(maxCount, currentCount);
                currentCount = 0;
            }
        }
        // Return the maximum of 'maxCount' and 'currentCount' in case the vector ends with ones
        return std::max(maxCount, currentCount);
    }
};
```

Typescript Solution

```typescript
/**
 * Finds the maximum number of consecutive 1's in an array.
 * @param {number[]} nums - The input array of numbers.
 * @return {number} The length of the longest sequence of consecutive 1's.
 */
function findMaxConsecutiveOnes(nums: number[]): number {
    let maxSequence = 0; // This variable will hold the maximum sequence length.
    let currentSequence = 0; // This variable will hold the current sequence length.

    // We iterate through each number in the array.
    for (const num of nums) {
        if (num === 0) {
            // If the current number is 0, we update the maxSequence if necessary,
            // and reset currentSequence since the sequence of 1's is broken.
            maxSequence = Math.max(maxSequence, currentSequence);
            currentSequence = 0;
        } else {
            // If the current number is 1, we increment the currentSequence counter.
            currentSequence++;
        }
    }

    // After the loop, we check one last time in case the array
    // ends with a sequence of 1's.
    return Math.max(maxSequence, currentSequence);
}
```

Time and Space Complexity

The time complexity of the given code is `O(n)`, where `n` is the length of the input list `nums`. This is because the code iterates over each element of `nums` exactly once, performing constant-time operations within the loop.

The space complexity of the code is `O(1)`. The only extra space used is for two integer variables `cnt` and `ans`, which are used to count the current streak of ones and store the maximum streak, respectively. Their space requirement does not vary with the size of the input list, so the space complexity is constant.