2229. Check if an Array Is Consecutive

Easy Array

Leetcode Link

Problem Description

The given problem presents us with a challenge to determine whether an array of integers, named `nums`, is consecutive or not. An array is considered consecutive if it includes all the numbers from `x` to `x + n - 1` without any gaps or repetitions, where `x` is the smallest number in the array, and `n` is the total count of numbers in the array. To reiterate, our task is to analyze the integer array and return `true` if the array meets this consecutive criteria or `false` otherwise.

Intuition

To intuitively solve this problem, we can break down the definition of a consecutive array into two main conditions:

1. The array must not contain any duplicate values, which is essential for ensuring that the array includes a complete range of numbers without repetition.
2. The difference between the maximum and minimum number in the array should be exactly `n - 1`, where `n` is the size of the array. This condition guarantees that the range spans enough numbers to fill the array without gaps.

So, the straightforward approach is to first identify the smallest and largest numbers in the array. Once these are identified, we can check whether all array elements are unique and whether the difference between the maximum and minimum values equals `n - 1`. If both conditions are satisfied, we can confidently determine that the array is consecutive.

Here's a step by step reasoning of the implementation:

- Find the minimum value `mi` in the array `nums` using the `min()` function. This represents `x`.
- Find the maximum value `mx` in the array `nums` using the `max()` function. This represents `x + n - 1`.
- Calculate the array size `n` using the `len()` function.
- Check if all elements in the array are unique by converting the array into a `set` and comparing its length to `n`.
- Finally, check if the maximum value equals the minimum value plus `n` minus 1, i.e., `mx == mi + n - 1`.
- If both checks pass, return `true`. Otherwise, return `false`.

By checking each of these conditions with simple operations, we can solve the problem in an efficient and understandable manner.

Solution Approach

The solution is implemented using simple Python constructs and relies on the unique properties of sets in Python, and mathematical reasoning. Here's the breakdown of the steps involved in checking if the list of integers is consecutive:

1. **Minimum and Maximum Values**: We begin by using Python's built-in `min()` and `max()` functions to find the smallest number `mi` and the largest number `mx` in the `nums` array. These functions iterate through the list and deliver the minimum and maximum values efficiently.

2. **Unique Elements Check**: We then convert the list `nums` into a set. Sets in Python are unordered collections of unique elements. By comparing the length of this set with the length of the original list (using `len(set(nums)) == n`), we determine if all elements in the array are unique. If there are any duplicates, the set's length will be smaller than that of the list because sets automatically remove duplicate items.

3. **Consecutive Numbers Check**: Finally, we check if the numbers are consecutive by verifying that the maximum element `mx` equals the minimum element `mi` plus `n - 1`. This check ensures that the numbers span a range which corresponds to the size of the list. If the length of the set is equal to the length of the list and the maximum value is as expected, the list is consecutive.

The solution code combines these three checks into one line:

```
return len(set(nums)) == n and mx == mi + n - 1
```

This line of code first asserts that each number in the array is unique (since the size of the set should be equal to the length of the array) and then ensures that the range (`mx - mi`) is exactly one less than the length of the array (`n - 1`). If both conditions are satisfied, this indicates that the array contains exactly `n` consecutive numbers starting at the minimum value found, thus making it a consecutive array, and the function returns `True`. Otherwise, the function returns `False`.

In terms of efficiency, this is a very concise approach since it involves a single pass over the array to check for all the required conditions, making it an O(n) time complexity algorithm, with O(n) space complexity due to set creation where `n` is the number of elements in the `nums` array.

Example Walkthrough

To illustrate the solution approach, consider a small example where the `nums` array is `[4, 5, 6, 7]`.

1. **Finding Minimum and Maximum Values**: First, the smallest number `mi` is found using `min(nums)`, which is `4` in this case. The largest number `mx` is found using `max(nums)`, which is `7` in our example.

2. **Unique Elements Check**: We convert the `nums` array into a set `{4, 5, 6, 7}`. The original list's length `n` is `4` and the set's length is also `4`. As the lengths are equal, we can conclude that all elements in the array are unique.

3. **Consecutive Numbers Check**: We now verify if the numbers are consecutive. We do this by checking if `mx = mi + n - 1`. In our example, `7 = 4 + 4 - 1` simplifies to `7 = 7`, which is true.

The array `[4, 5, 6, 7]` passed the unique elements check and the consecutive numbers check. Thus, by implementing the solution approach:

```
mi = min(nums)
mx = max(nums)
n = len(nums)
return len(set(nums)) == n and mx == mi + n - 1
```

The conditions are satisfied and the function would return `True`, confirming that the array is consecutive.

Python Solution

```python
class Solution:
    def isConsecutive(self, nums: List[int]) -> bool:
        # Find the minimum and maximum value in the list
        min_value = min(nums)
        max_value = max(nums)
        # Get the length of the list
        num_length = len(nums)

        # To be a consecutive sequence, two conditions must be met:
        # 1. All numbers must be unique (no duplicates) which is checked by converting
        #    the list to a set and comparing its length with the original list length.
        # 2. The difference between max and min value should be exactly one less than
        #    the length of the list (since a consecutive sequence increments by one
        #    for each element).
        # Therefore, we return True if both conditions are met, False otherwise.
        return len(set(nums)) == num_length and max_value == min_value + num_length - 1
```

Java Solution

```java
class Solution {
    public boolean isConsecutive(int[] nums) {
        int minVal = nums[0];               // Initialize minimum value with the first element.
        int maxVal = nums[0];               // Initialize maximum value with the first element.
        Set<Integer> seenNumbers = new HashSet<>();  // Create a set to store unique numbers.

        for (int value : nums) {
            minVal = Math.min(minVal, value);       // Update the minimum value.
            maxVal = Math.max(maxVal, value);       // Update the maximum value.
            seenNumbers.add(value);                 // Add the current value to the set.
        }

        int length = nums.length;           // Store the length of the input array.

        // Verify two conditions for the array to consist of consecutive elements:
        // 1. The set size should be equal to the array length (no duplicates).
        // 2. The maximum value should equal the minimum value plus the array length minus 1.
        return seenNumbers.size() == length && maxVal == minVal + length - 1;
    }
}
```

C++ Solution

```cpp
#include <vector>
#include <unordered_set>
#include <algorithm>

class Solution {
public:
    bool isConsecutive(std::vector<int>& nums) {
        // Create a set containing all elements from the vector.
        // Sets automatically remove duplicates.
        std::unordered_set<int> elementsSet(nums.begin(), nums.end());

        // Find the smallest element in the vector.
        int minElement = *std::min_element(nums.begin(), nums.end());

        // Find the largest element in the vector.
        int maxElement = *std::max_element(nums.begin(), nums.end());

        // Get the count of unique numbers in the vector.
        int numCount = nums.size();

        // Check if there are no duplicates (set size == vector size)
        // and if the range between min and max elements is exactly the size of the vector - 1.
        // This ensures that the vector contains consecutive numbers.
        return elementsSet.size() == numCount && maxElement == minElement + numCount - 1;
    }
};
```

Typescript Solution

```typescript
// Imports not required in TypeScript for array operations

// Function to check if an array consists of consecutive integers.
function isConsecutive(nums: number[]): boolean {
    // Convert the array into a set to filter out duplicate elements.
    const elementsSet: Set<number> = new Set(nums);

    // Find the smallest element in the array.
    const minElement: number = Math.min(...nums);

    // Find the largest element in the array.
    const maxElement: number = Math.max(...nums);

    // Count the number of unique elements in the array.
    const numCount: number = nums.length;

    // Check if there are no duplicates (set size equals array size) and
    // if the range between the smallest and largest elements equals to the array length - 1.
    // This condition confirms the array contains consecutive numbers.
    return elementsSet.size === numCount && maxElement === minElement + numCount - 1;
}

// Example usage:
// const result = isConsecutive([1, 2, 3, 4, 5]);
// console.log(result); // Output should be true if the input array is consecutive.
```

Time and Space Complexity

The time complexity of the code is `O(n)` where `n` is the length of the `nums` list. This is because the `min()` and `max()` functions will each take `O(n)` to find the minimum and maximum elements in the list, and `len()` takes `O(1)` time. Furthermore, converting the list to a set also takes `O(n)` time in average case to eliminate duplicates and check if all the elements are unique.

The space complexity of the code is `O(n)` since we are creating a new set from the list of numbers, which in the worst case will contain a unique value for every element in the list if there are no duplicates.