

961. N-Repeated Element in Size 2N Array

EasyArrayHash Table

Problem Description

You are provided with an integer array, `nums`, which has some unique properties. The length of the array is `2n`, indicating that it consists of twice the quantity of a certain number `n`. Within this array, there are `n + 1` distinct elements, meaning there are only a few unique elements. Among these unique elements, there is one particular element that appears exactly `n` times. The challenge here is to identify that one element which is repeated `n` times and return it.

Intuition

The intuition behind the solution lies in understanding the properties of the set data structure and the constraints mentioned in the problem. A set in Python is a collection that is unordered and unindexed, and most importantly, it does not allow duplicate elements. Since we know there are `n` repeats of the same element and only `n + 1` unique elements in the array, we can iterate through the array and add each element to the set.

Solution Approach

The implementation of the solution directly follows the intuition. Here's a step-by-step walk-through of the implementation, which is quite simple and efficient due to the particular constraints of the problem:

- A set, `s`, is created to keep track of the unique elements we encounter in the `nums` array. In Python, a set is valuable because it stores elements in an unordered fashion and, most importantly for our case, does not allow duplicates.
- We iterate over each element `x` in the array `nums`. For each element, we perform a check to determine whether it already exists in the set `s`.
- If `x` is not present in the set (which means this is the first time we're seeing this number), we add it to the set using `s.add(x)`.
- If `x` is already in the set `s`, then it means we have found the duplicate element which has occurred `n` times, in compliance with the problem's constraints. We then immediately return `x`.
- The loop will continue until the condition in step 4 is met, which is guaranteed to happen since we know from the problem statement that one element is repeated `n` times.

By using the set and leveraging its properties, we avoid having to compare each element with every other element, which would be a less efficient approach. The use of a set provides a quick and concise way to detect the repeat without additional memory for counting or additional nested loops, making this algorithm run in $O(n)$ time complexity and $O(n)$ space complexity, where n is the number of unique elements in the array.

Example Walkthrough

Let's illustrate the solution approach with a small example. Suppose we are given an array `nums` with the following elements: `[1, 2, 3, 3]`.

Following the step-by-step solution:

- First, we create an empty set `s` to store unique elements we encounter in the `nums` array. Initially, `s = {}`.
- We start iterating over each element `x` in the `nums` array.
 - On the first iteration, `x = 1`. Since `1` is not in the set `s`, we add `1` to `s`. Now `s = {1}`.
 - On the second iteration, `x = 2`. Since `2` is not in `s`, we add `2` to `s`. Now `s = {1, 2}`.
 - On the third iteration, `x = 3`. Since `3` is not in `s`, we add `3` to `s`. Now `s = {1, 2, 3}`.
 - On the fourth iteration, `x = 3` again. But this time, `3` is already in `s`, so we have found our duplicate element.
- As soon as we find that `3` is already in the set `s`, we don't need to look any further. We immediately return `3` as the duplicate element that is repeated `n` (2) times.

This example showed how the algorithm efficiently traverses the array and uses a set to find the repeating element without the need for unnecessary comparisons or additional memory for counts. The implementation matches the intuition and the problem constraints, offering a simple yet powerful solution.

Solution Implementation

Python

```
from typing import List

class Solution:
    def repeatedNTimes(self, nums: List[int]) -> int:
        """
        Find the element that is repeated N times in the given list, where the list size is 2N.

        Args:
            nums: List[int] - A list of integers with size 2N.

        Returns:
            int - The integer that is repeated N times.
        """

        seen = set() # Initialize an empty set to keep track of seen elements.

        for num in nums:
            if num in seen:
                # If the number is already in the set, we have found our repeated element.
                return num
            # Add the current number to the set.
            seen.add(num)

        # The function should always return within the loop for well-formed inputs.
        # An explicit return None could be added here, but it's unnecessary.
```

Java

```
class Solution {
    // This method finds the element that is repeated N times in the array `nums`.
    public int repeatedNTimes(int[] nums) {
        // Create a HashSet to store unique elements.
        // The initial capacity is set to nums.length / 2 + 1 because we know there is
        // one element repeating N times in a 2N sized array.
        Set<Integer> uniqueElements = new HashSet<>(nums.length / 2 + 1);

        // Iterate through each element in the array.
        for (int i = 0;; ++i) { // the loop will break from the inside so no condition is set here
            // Attempt to add the current element to the HashSet.
            // If the add() method returns false, it means the element is already in the set
            // and hence, it is the element that repeats N times.
            if (!uniqueElements.add(nums[i])) {
                // Return the repeated element.
                return nums[i];
            }
        }
        // Since there must be an N times repeated element by the problem statement, the loop
        // is guaranteed to return at some point and does not require an explicit termination condition.
    }
}
```

C++

```
#include <vector>
#include <unordered_set>

class Solution {
public:
    // Function to find the element that is repeated N times in the array
    int repeatedNTimes(vector<int>& nums) {
        // Create an unordered set to keep track of visited elements
        unordered_set<int> visitedElements;

        // Iterate over the array
        for (int i = 0; i < nums.size(); ++i) {
            // Check if the current element is already in the set
            if (visitedElements.count(nums[i])) {
                // If it is in the set, we've found the repeated element
                return nums[i];
            }
            // If it is not in the set, add the current element to the set
            visitedElements.insert(nums[i]);
        }

        // If the loop completes without returning, there is a problem with the input
        // as the problem definition guarantees a repeated element
        // However, the return statement below is needed to avoid a compiler error.
        return -1;
    }
};
```

TypeScript

```
/**
 * Finds the element that is repeated N times in an array where all other elements appear exactly once.
 *
 * @param {number[]} elements - An array of numbers which contains a unique element and one element repeated N times.
 * @returns {number} - The number that is repeated N times.
 */
function repeatedNTimes(elements: number[]): number {
    // Initialize a new Set to store unique elements as we encounter them.
    const uniqueElements: Set<number> = new Set();

    // Iterate through each number in the array.
    for (const element of elements) {
        // Check if the current element is already in the Set.
        if (uniqueElements.has(element)) {
            // If so, we have found our repeated element and return it.
            return element;
        }
        // Otherwise, add the element to the Set of unique elements.
        uniqueElements.add(element);
    }

    // The function assumes that there will always be a repeated element,
    // so there is no explicit return statement outside the loop.
    // TypeScript will infer that the end of the function is unreachable code.
    // If the input is guaranteed to always have a repeated element, this is fine.
    // Otherwise, you should handle the case where no element is repeated:
    // throw new Error('No repeated element found');
}
```

```
from typing import List

class Solution:
    def repeatedNTimes(self, nums: List[int]) -> int:
        """
        Find the element that is repeated N times in the given list, where the list size is 2N.

        Args:
            nums: List[int] - A list of integers with size 2N.

        Returns:
            int - The integer that is repeated N times.
        """

        seen = set() # Initialize an empty set to keep track of seen elements.

        for num in nums:
            if num in seen:
                # If the number is already in the set, we have found our repeated element.
                return num
            # Add the current number to the set.
            seen.add(num)

        # The function should always return within the loop for well-formed inputs.
        # An explicit return None could be added here, but it's unnecessary.
```

Time and Space Complexity

The given Python code aims to find an element that is repeated `N` times in a list where `2N` elements are present, and `N-1` elements are unique.

Time Complexity

The time complexity is $O(N)$ because the function iterates through each element of the list exactly once in the worst case (where N is the length of the list).

Space Complexity

The space complexity is also $O(N)$ as it uses a set to store elements encountered in the list. In the worst case, this set will store $N-1$ elements when the repeated element is the last one to be checked.