349. Intersection of Two Arrays

Easy | Array | Hash Table | Two Pointers | Binary Search | Sorting

Leetcode Link

Problem Description

The problem requires us to find the intersection of two integer arrays, `nums1` and `nums2`. The intersection of two arrays means the elements that are common to both arrays. However, there are some special rules in this problem:

- Each element in the resulting array must be unique, which means if an element appears more than once in the intersection, it should only appear once in the result.
- The result can be returned in any order, so there is no need to sort the result or follow the order of any of the input arrays.

The goal is to write a function that takes these two arrays as input and returns a new array that satisfies these conditions.

Intuition

To arrive at the solution, we need to think about the most efficient way to find common elements between the two given arrays. Here are a few key points to consider:

1. Since each element in the result must be unique, using a set data structure is a natural choice because sets automatically remove duplicate elements.
2. To find the common elements, we can convert both arrays into sets and then find the intersection of these sets. The intersection operation is well defined in mathematics and programming languages as the set of elements that are present in both sets.
3. The intersection operator in Python (denoted by `&`) conveniently does this job for us, returning another set that contains only the elements that are found in both `set(nums1)` and `set(nums2)`.
4. After we get the intersection, its elements are unique by the definition of a set. However, since the final output requires the result to be a list, we convert this intersection set back to a list using the `list()` function.

The combination of these steps provides us with a simple and elegant solution to the problem, which is both easy to code and understand.

Solution Approach

The implementation of the solution uses a set intersection approach, leveraging Python's built-in set operations. Here is a step-by-step walkthrough of the algorithm and the associated data structures or patterns used:

1. **Convert lists to sets**: The first step is to convert the two input lists `nums1` and `nums2` into sets. This is done using the `set()` function in Python. The purpose of this conversion is two-fold: to eliminate any duplicate elements within each array and to prepare the data for set-based operations.

```
set_nums1 = set(nums1)
set_nums2 = set(nums2)
```

2. **Intersect the sets**: Once we have two sets, we can find their intersection. In Python, the intersection of two sets can be found using the `&` operator. This operation will return a new set containing only the elements that are present in both `set_nums1` and `set_nums2`.

```
intersection_set = set_nums1 & set_nums2
```

3. **Convert the set back to a list**: After finding the intersection, we have a set of unique elements that are present in both input arrays. Since the problem requires the result to be returned as a list, the final step is to convert this set back into a list. We can do this simply by passing the set to the `list()` constructor.

```
result_list = list(intersection_set)
```

4. **Return the result**: The result now contains all the unique elements that are present in both `nums1` and `nums2`. This list is returned as the final output of the function.

By succinctly combining these steps within the `intersection` method of the `Solution` class, the entire process is captured in a single, readable line of code:

```
return list(set(nums1) & set(nums2))
```

This one-liner showcases the power and simplicity of Python for solving such problems, where the language's built-in data structures and operators do much of the heavy lifting for us.

Example Walkthrough

Let's illustrate the solution approach with a small example.

Suppose we have two integer arrays: `nums1 = [1, 2, 2, 1]` and `nums2 = [2, 2, 3]`.

- **Convert lists to sets**: As per the first step, we convert these lists into sets to eliminate any duplicates.

```
set_nums1 = set([1, 2, 2, 1]) => {1, 2}
set_nums2 = set([2, 2, 3]) => {2, 3}
```

- **Intersect the sets**: Now, we find the intersection of these two sets to identify the common elements. The `&` operator helps us with this.

```
intersection_set = {1, 2} & {2, 3} => {2}
```

After this step, `intersection_set` holds the unique elements that are present in both `set_nums1` and `set_nums2`, which in this case is the single element `{2}`.

- **Convert the set back to a list**: We then convert the resulting set back into a list since the function is expected to return a list.

```
result_list = list({2}) => [2]
```

- **Return the result**: The final resulting list `[2]` contains all the unique intersection elements from both `nums1` and `nums2`. This is the output we would return from the function.

By applying the solution approach to this example, we successfully identified the intersection of two arrays with duplicate elements, reduced them to their unique elements, and provided the required list output. The elegance of Python's set operations made this process straightforward and efficient. Applying this same approach to the question at hand, the function returns the list `[2]` as the intersection between `nums1` and `nums2`.

Python Solution

```
from typing import List

class Solution:
    def intersection(self, nums1: List[int], nums2: List[int]) -> List[int]:
        # Convert the first list to a set to eliminate duplicates
        set_nums1 = set(nums1)

        # Convert the second list to a set to eliminate duplicates
        set_nums2 = set(nums2)

        # Find the intersection of the two sets, which contains only the common elements
        intersection_set = set_nums1 & set_nums2

        # Convert the resulting set to a list before returning
        return list(intersection_set)

# The code defines a method intersection within the Solution class that takes two lists of integers
# and returns a list of the unique elements that are present in both lists.
```

Java Solution

```
import java.util.ArrayList;
import java.util.List;

class Solution {
    // Finds the intersection of two arrays, i.e., elements that appear in both arrays.
    public int[] intersection(int[] nums1, int[] nums2) {
        // Initialize a boolean array to track which elements from nums1 have been seen.
        boolean[] seen = new boolean[1001]; // Assumes the elements are in the range [0, 1000].

        // Mark elements present in nums1 as seen.
        for (int num : nums1) {
            seen[num] = true;
        }

        // Use a list to collect the intersection elements.
        List<Integer> intersectionElements = new ArrayList<>();
        // Iterate over nums2 to find common elements.
        for (int num : nums2) {
            // If an element from nums2 has been seen and is not yet included in the intersectionElements.
            if (seen[num]) {
                intersectionElements.add(num); // Add the element to the intersection list.
                seen[num] = false; // Mark it as not seen to avoid duplicates in the intersectionElements.
            }
        }

        // Convert the List of Integer to an array of int for the result.
        int[] result = new int[intersectionElements.size()];
        for (int i = 0; i < intersectionElements.size(); i++) {
            result[i] = intersectionElements.get(i);
        }

        // Return the result array containing the intersection of the two arrays.
        return result;
    }
}
```

C++ Solution

```
#include <vector>
#include <cstring> // Required for memset function

class Solution {
public:
    // Computes the intersection of two vectors, nums1 and nums2,
    // and returns the unique elements present in both.
    std::vector<int> intersection(std::vector<int>& nums1, std::vector<int>& nums2) {
        // Create an array to mark the presence of elements.
        // Assuming the elements in nums1 and nums2 are in the range [0, 1000]
        bool seen[1001];
        std::memset(seen, false, sizeof(seen)); // Initialize all elements in 'seen' to 'false'

        // Mark all elements present in nums1 in the 'seen' array
        for (int num : nums1) {
            seen[num] = true;
        }

        // Vector to store the intersection result
        std::vector<int> result;

        // Iterate through nums2 to find common elements,
        // add them to result and mark them as 'false' in 'seen' to avoid duplicates
        for (int num : nums2) {
            if (seen[num]) {
                result.push_back(num); // If element is seen, add to result
                seen[num] = false;     // Set to 'false' to avoid adding duplicates
            }
        }

        // Return the vector containing the intersection of nums1 and nums2
        return result;
    }
};
```

Typescript Solution

```
/**
 * Finds the intersection of two arrays, meaning the elements that are present in both arrays.
 * The function assumes the numbers in both arrays are integers and within the range of 0 to 1000.
 *
 * @param {number[]} nums1 - First array of numbers.
 * @param {number[]} nums2 - Second array of numbers.
 * @return {number[]} - Array containing the intersection of both input arrays.
 */
const intersection = (nums1: number[], nums2: number[]): number[] => {
    // Initialize a boolean array with a fixed size of 1001,
    // corresponding to the specified range of numbers.
    const seen: boolean[] = new Array(1001).fill(false);

    // Mark the numbers that are present in the first array as true in the 'seen' array.
    for (const number of nums1) {
        seen[number] = true;
    }

    // Initialize the array that will hold the intersection result.
    const result: number[] = [];

    // Traverse the second array and check if the current number is seen.
    // If it is, add it to the result and set its 'seen' value to false
    // to prevent duplicates if it appears again in nums2.
    for (const number of nums2) {
        if (seen[number]) {
            result.push(number);
            seen[number] = false;
        }
    }

    // Return the resultant array containing the intersection of nums1 and nums2.
    return result;
};

// Example usage:
// const nums1 = [1, 2, 2, 1];
// const nums2 = [2, 2];
// const result = intersection(nums1, nums2); // result would be [2]
```

Time and Space Complexity

Time Complexity

The time complexity for the given solution involves two main operations: converting the lists to sets and finding the intersection of these sets.

1. Converting `nums1` to a set: This operation has a time complexity of $O(n)$ where n is the number of elements in `nums1`.
2. Converting `nums2` to a set: Similarly, this has a time complexity of $O(m)$ where m is the number of elements in `nums2`.
3. Finding the intersection: The intersection operation `&` for sets is normally $O(min(n, m))$ because it checks each element in the smaller set for presence in the larger set.

Thus, the overall time complexity can be summarized as $O(max(n, m))$ assuming that the intersection operation is dominated by the process of converting the lists to sets.

Space Complexity

The space complexity considers the additional space required besides the input:

1. Space for the set of `nums1`: This is $O(n)$.
2. Space for the set of `nums2`: This is $O(m)$.

Since these sets do not depend on each other, the total space complexity is $O(n + m)$ for storing both sets.