Documentation: Intersection of Two Arrays II

The problem of finding the intersection of two arrays is a classic computational challenge that involves determining the common elements between two given lists. The result must account for the frequency of elements, meaning each number should appear in the result as many times as it appears in both arrays. This introduces additional complexity compared to simply finding unique intersecting elements. Solving this problem efficiently is crucial, especially when working with larger datasets.

Intuition

At its core, the task revolves around counting and comparing the occurrences of each element in the two input arrays. The intersection of the arrays can be determined by taking the minimum frequency of each common element. This approach ensures that the result respects the constraints of preserving the frequency of elements. Using a dictionary-like data structure to count occurrences makes the process intuitive and manageable, as it allows us to track the frequency of elements in each array efficiently.

Approach

The first step is to count the occurrences of elements in both arrays. Using a Counter from Python's collections module simplifies this, as it automatically generates a frequency dictionary for a list. Once the counts are available for both arrays, we loop through the elements of the smaller Counter and check for their presence in the other Counter. For elements that exist in both arrays, the minimum frequency from the two counters determines how many times the element will appear in the result. The result is then built dynamically by appending the intersecting elements to a list.

Complexity Analysis

The time complexity of the solution is primarily driven by the steps to count the frequencies of the arrays and compute the intersection. Counting the elements in both arrays takes O(n + m), where n and m are the lengths of the arrays. Iterating through the unique elements in one of the Counter objects (the smaller one) adds O(k), where k is the number of unique elements in the smaller array. Thus, the overall time complexity is O(n + m + k). The space complexity is O(u1 + u2 + p), where u1u1 and u2 are the numbers of unique elements in nums1 and nums2, and pp is the size of the result array.

Practical Examples

For example, given two arrays [1, 2, 2, 1] and [2, 2], their intersection is [2, 2] because the number 2 appears twice in both arrays. Another example is [4, 9, 5] and [9, 4, 9, 8, 4], where the intersection could be [4, 9] or [9, 4], as the result order does not matter. These examples highlight the importance of considering the frequency of elements while constructing the intersection.

Handling Edge Cases

Edge cases include scenarios where one or both arrays are empty and the intersection is also empty. Another scenario is when there are no common elements between the two arrays, resulting in an empty intersection. Additionally, if one array is significantly larger than the other, efficiency becomes a priority, and selecting the smaller variety for the Counter ensures that memory and computation are optimized.

Follow-Up Questions

Several follow-up scenarios challenge the robustness of the solution. For sorted arrays, a two-pointer technique can be applied for O(n + m) time and O(1) space, making it optimal for such cases. If one array is much smaller than the other, using a hash map for the smaller array minimizes space and computational overhead. When dealing with large datasets where one array (e.g., nums2) cannot fit in memory, a streaming approach is ideal, processing chunks of nums2 at a time to compute the intersection incrementally.

Conclusion

This problem is a great example of leveraging data structures like Counter to solve frequency-based problems effectively. It also highlights the importance of adapting algorithms to specific constraints, such as sorted arrays, unevenly sized inputs, or memory limitations. The structured approach to analyzing, designing, and optimizing the solution ensures both clarity and efficiency, making it adaptable to real-world applications.