Documentation

Intuition

The RandomizedCollection is a specialized data structure designed to efficiently manage a multiset of integers, where duplicates are allowed. The primary operations include inserting, removing, and retrieving random values with uniform probability. Achieving O(1) average time complexity for all these operations requires careful use of data structures that complement each other in performance and flexibility.

The key idea is to combine a **list** and a **dictionary**. The list (elements) stores the actual values of the multiset, while the dictionary (indices) maps each value to a set of indices where the value is located in the list. This combination allows for efficient updates to both data structures during insertion and removal, while maintaining the uniform random property for the getRandom operation.

Approach

To implement the RandomizedCollection, three main operations need to be addressed. For the **insert** operation, the value is added to the list, and its index is recorded in the dictionary. If the value was not previously in the collection, the operation returns True, otherwise False. This ensures we track the first occurrence accurately. For the **remove** operation, a single occurrence of the value is removed. This involves swapping the value with the last element in the list and then updating the dictionary accordingly. The **getRandom** operation retrieves a random element by directly accessing a random index in the list, which guarantees uniform selection.

A significant challenge in the approach is efficiently handling duplicates during the removal process. By using a set in the dictionary to track indices, duplicates can coexist without overwriting each other. This ensures that the structure maintains correctness while allowing O(1) operations.

Insert Operation

The insert operation checks if the value exists in the dictionary. If not, it initializes a new entry. Regardless of whether it exists, the value is appended to the list, and its index is recorded in the dictionary. This guarantees O(1) insertion time. Returning True for the first occurrence and False for subsequent ones aligns with the requirements of distinguishing between first-time and repeated insertions.

Remove Operation

The removal operation is more complex due to the need to manage duplicates and maintain list integrity. If the value does not exist in the dictionary or has no recorded indices, the operation immediately returns False. Otherwise, one of its indices is removed. If the removed index is not the last in the list, the last element is swapped into the removed index's position, and the dictionary is updated to reflect this change. Finally, the last element is removed from the list. These steps ensure efficient removal without disrupting the O(1) complexity.

GetRandom Operation

The getRandom operation leverages Python's random.choice, which directly accesses a random index in the list. This operation is O(1) since no additional computation is required beyond generating a random index. The uniform probability of selection is maintained as all elements are equally likely to be chosen due to their presence in the list.

Complexity Analysis

The average time complexity for all operations is O(1). Insertions and removals achieve this by only manipulating the end of the list and updating the dictionary. The getRandom operation is inherently O(1) due to direct access to list elements. Space complexity is O(n), where n is the total number of elements in the collection. This accounts for storage in both the list and the dictionary.

Applications

The RandomizedCollection is particularly useful in scenarios where a dynamic collection of elements needs to support frequent updates and random access. Potential applications include gaming systems, simulations, and randomized algorithms. Its ability to handle duplicates efficiently adds versatility, making it suitable for a broader range of use cases compared to simpler data structures like sets.