Problem Name: Contains Duplicate II

Link: https://leetcode.com/problems/contains-duplicate-ii/

Solution Language(s): Java

Approach

In this problem we are tasked with determining if there are any duplicate numbers in the given list *nums* that are less than or equal to *k* indexes away from one another. My approach was to search through the array *nums* and for each value, add the value and index to a HashMap if it did not already exist. If it did, I would check if the value (index) associated with the key (the duplicate integer), was less than or equal to *k* indices away from the current duplicate. Essentially, I would check if the most recent occurrence of the duplicate was less than or equal to *k* indices away from the last occurrence of that duplicate. If not, the value associated with the duplicate integer, the index, would be updated to the most recent occurrence. If the loop iterates all the way through, then there are no duplicates that meet the constraints.

Solution

```java
class Solution {
    public boolean containsNearbyDuplicate(int[] nums, int k) {
        HashMap<Integer, Integer> dup = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            if (dup.containsKey(nums[i]) &&
            Math.abs(dup.get(nums[i]) - i) <= k) {
                return true;
            } else {
                dup.put(nums[i], i);
            }
        }
        return false;
    }
}
```

Time Complexity: O(n)

The time complexity in this approach is O(n), where *n* is the number of elements in *nums,* as we may have to search through the entire list if there are no duplicates within *k* indices apart.

Space Complexity: O(n)

The space complexity of this method is O(n) because we may end up adding all the elements from the array *nums* to the HashMap.

Conclusion

In conclusion, this problem is relatively simple, and contains two main solutions. Mine is a bit more obscure than the obvious solution which is to go through the array *nums,* with a sliding window approach. This involves having a HashSet which iterates through the array *nums* until it becomes larger than *k.* Before each add it would check if you are adding a duplicate, in which case it would return true, that there is a duplicate within *k* indices apart. Otherwise, you would simply add the value to the HashSet. At the point where the HashSet becomes larger than *k,* you would remove the leftmost value in the HashMap. You would then continually add to the HashMap and then remove in a cycle until you reached the end of the array *nums.*

Alternate Approach

```java
class Solution {
    public boolean containsNearbyDuplicate(int[] nums, int k) {
        HashSet<Integer> set = new HashSet<>();
        for (int i = 0; i < nums.length; i++) {
            if (set.contains(nums[i])) {
                return true;
            }
            set.add(nums[i]);
            if (set.size() > k) {
                set.remove(nums[i - k]);
            }
        }
        return false;
    }
}
```