<https://leetcode.com/problems/subarrays-with-k-different-integers/description/>

Given an integer array nums and an integer k, return the number of **good subarrays** of nums.

A **good array** is an array where the number of different integers in that array is exactly k.

For example, [1,2,3,1,2] has 3 different integers: 1, 2, and 3.

A **subarray** is a **contiguous** part of an array.

**Example 1:**

**Input:** nums = [1,2,1,2,3], k = 2

**Output:** 7

**Explanation:** Subarrays formed with exactly 2 different integers: [1,2], [2,1], [1,2], [2,3], [1,2,1], [2,1,2], [1,2,1,2]

**Example 2:**

**Input:** nums = [1,2,1,3,4], k = 3

**Output:** 3

**Explanation:** Subarrays formed with exactly 3 different integers: [1,2,1,3], [2,1,3], [1,3,4].

**Constraints:**

1 <= nums.length <= 2 \* 10^4

1 <= nums[i], k <= nums.length

**Attempt 1: 2025-01-04**

**Solution 1: Not fixed length Sliding Window (10 min)**

**The biggest difference between L992 and** [L904.P2.4.Fruit Into Baskets (Ref.L340)](note://5BC9A29D8C844ABA8C42A0DE6FB89BAC) **is L904 try to get at most k distinct integer on subarray, L992 try to get exactly k distinct integer on subarray, so the logic for L992 requires L904 logic on k minus L904 logic on k - 1 to get exactly k distinct integer result**

class Solution {

    public int subarraysWithKDistinct(int[] nums, int k) {

        return helper(nums, k) - helper(nums, k - 1);

    }

    // 'helper' method exactly same as L904

    private int helper(int[] nums, int k) {

        int count = 0;

        Map<Integer, Integer> freq = new HashMap<>();

        int i = 0;

        for(int j = 0; j < nums.length; j++) {

            freq.put(nums[j], freq.getOrDefault(nums[j], 0) + 1);

            while(freq.size() > k) {

                freq.put(nums[i], freq.get(nums[i]) - 1);

                if(freq.get(nums[i]) == 0) {

                    freq.remove(nums[i]);

                }

                i++;

            }

            count += (j - i + 1);

        }

        return count;

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to chatGPT**

LeetCode 992 ("Subarrays with K Different Integers") requires finding the number of subarrays with exactly K distinct integers. A common approach to solve this problem is using the sliding window technique with the help of a helper function to calculate the number of subarrays with at most K distinct integers. The difference between results of "at most K" and "at most K-1" gives the result for "exactly K".

Here's an efficient Java solution:

public class Solution {

public int subarraysWithKDistinct(int[] nums, int k) {

return atMostKDistinct(nums, k) - atMostKDistinct(nums, k - 1);

}

private int atMostKDistinct(int[] nums, int k) {

int left = 0, count = 0;

HashMap<Integer, Integer> freqMap = new HashMap<>();

for (int right = 0; right < nums.length; right++) {

// Add the current number to the frequency map

freqMap.put(nums[right], freqMap.getOrDefault(nums[right], 0) + 1);

// If there are more than k distinct numbers, shrink the window

while (freqMap.size() > k) {

freqMap.put(nums[left], freqMap.get(nums[left]) - 1);

if (freqMap.get(nums[left]) == 0) {

freqMap.remove(nums[left]);

}

left++;

}

// Add the count of subarrays ending at 'right'

count += right - left + 1;

}

return count;

}

}

**Explanation**

atMostKDistinct **Helper Function**:

Keeps track of the number of subarrays with at most K distinct integers.

Uses a sliding window approach with two pointers (left and right).

Maintains a freqMap to store the frequency of integers in the current window.

If the number of distinct integers exceeds K, the window is shrunk from the left.

subarraysWithKDistinct **Main Function**:

The number of subarrays with exactly K distinct integers is the difference between the number of subarrays with at most K and the number of subarrays with at most K-1.

**Complexity**

Time Complexity: O(n) where n is the length of the array. Each element is processed at most twice (once when expanding and once when shrinking the window).

Space Complexity: O(k) for the frequency map storing at most k distinct elements.

**The atMostKDistinct() method similar to which Leetcode problem ?**

The atMostKDistinct() method is similar to the logic used in **LeetCode 904 - Fruit Into Baskets**.

**Similarity to LeetCode 904**

In **LeetCode 904**, the goal is to find the longest subarray (contiguous segment) of an array with at most 2 distinct integers.

The solution involves a sliding window with two pointers (left and right) and a frequency map to track the count of elements within the window. This is effectively the same logic as the atMostKDistinct() method, but with k=2k = 2k=2.

Here’s the similarity in approach:

Maintain a frequency map (or similar structure) to count the occurrences of integers within the current window.

Expand the window by moving the right pointer.

Shrink the window by moving the left pointer if the number of distinct integers exceeds the allowed limit (K or 2 in the case of LeetCode 904).

**Example Code for LeetCode 904**

Here’s how Fruit Into Baskets is solved:

import java.util.HashMap;

public class Solution {

public int totalFruit(int[] fruits) {

int left = 0, maxLength = 0;

HashMap<Integer, Integer> freqMap = new HashMap<>();

for (int right = 0; right < fruits.length; right++) {

// Add current fruit to the frequency map

freqMap.put(fruits[right], freqMap.getOrDefault(fruits[right], 0) + 1);

// Shrink the window if there are more than 2 distinct fruits

while (freqMap.size() > 2) {

freqMap.put(fruits[left], freqMap.get(fruits[left]) - 1);

if (freqMap.get(fruits[left]) == 0) {

freqMap.remove(fruits[left]);

}

left++;

}

// Update the maximum length

maxLength = Math.max(maxLength, right - left + 1);

}

return maxLength;

}

}

**Differences**

LeetCode 904 focuses on finding the longest subarray with at most 2 distinct integers, so it directly returns the length of the longest window.

In **LeetCode 992**, we use the same logic to count the number of subarrays with at most K distinct integers, which is then used to calculate the subarrays with exactly K distinct integers.

**Refer to**

[L904.P2.4.Fruit Into Baskets (Ref.L340)](note://5BC9A29D8C844ABA8C42A0DE6FB89BAC)

[L2062.Count Vowel Substrings of a String (Ref.L2461)](note://WEBffe1facb0ba61c1f4e6681e23ad781cb)

[L2461.Maximum Sum of Distinct Subarrays With Length K (Ref.L2062,L2401)](note://WEB7bdebd65515808a4d25cc7c45804eb07)