

# Editorial: Diverse Substring Sum

## Problem Statement

Given a string  $S$  of length  $n$  and an integer  $k$ , count the number of substrings of  $S$  that are *diverse*. A substring is diverse if it contains at most  $k$  distinct characters. The solution must handle cases efficiently for both small and large  $k$ .

## Core Observation

The key insight is recognizing the constraint on the alphabet size. In typical problem constraints (e.g.,  $n \leq 10^5$ ), the alphabet size is implicitly bounded by 20 for efficient computation. Crucially:

- If  $k > 20$ , then  $2^k > n$  holds (since  $2^{20} = 1,048,576 > 10^5 \geq n$ ). More importantly, the maximum number of distinct characters in *any* substring cannot exceed the alphabet size, which is at most 20. Thus, for  $k > 20$ , every substring has  $\leq 20 < k$  distinct characters, making all substrings diverse.
- If  $k \leq 20$ , the alphabet size constraint allows an efficient  $O(n)$  sliding window approach to count valid substrings.

This dichotomy leverages the small alphabet size to handle large  $k$  trivially while using a targeted method for small  $k$ .

## Solution

The solution branches based on the value of  $k$ :

### Case 1: $k > 20$

As established, all substrings are diverse because the alphabet size  $\leq 20 < k$ . The total number of substrings of a string of length  $n$  is  $\frac{n(n+1)}{2}$ . Thus, the answer is:

$$\boxed{\frac{n(n+1)}{2}}$$

### Case 2: $k \leq 20$

We use a sliding window (two-pointer) technique with a frequency array to count substrings with at most  $k$  distinct characters in  $O(n)$  time. The algorithm maintains a window  $[l, r]$  where the substring  $S[l..r]$  has  $\leq k$  distinct characters.

1. Initialize:

- Pointers  $l = 0, r = 0$
- Frequency array  $\text{freq}[1..26]$  (for 'a'-'z') initialized to 0

- $\text{distinct} = 0$  (current distinct characters in window)
  - $\text{count} = 0$  (accumulator for valid substrings)
2. Iterate  $r$  from 0 to  $n - 1$ :
    - (a) Increment  $\text{freq}[S[r]]$ . If  $\text{freq}[S[r]] = 1$ , increment  $\text{distinct}$ .
    - (b) While  $\text{distinct} > k$ :
      - Decrement  $\text{freq}[S[l]]$ . If  $\text{freq}[S[l]] = 0$ , decrement  $\text{distinct}$ .
      - Increment  $l$ .
    - (c) All substrings ending at  $r$  and starting in  $[l, r]$  are valid. Add  $(r - l + 1)$  to  $\text{count}$ .
  3. Output  $\text{count}$ .

**Why this works:** For each  $r$ , the window  $[l, r]$  is the longest valid window ending at  $r$ . The number of valid substrings ending at  $r$  is exactly the window length  $(r - l + 1)$ , as shrinking the window from the left maintains validity.

## Complexity Analysis

- **Time Complexity:**
  - For  $k > 20$ :  $O(1)$  (direct formula computation).
  - For  $k \leq 20$ :  $O(n)$ . Each character is processed at most twice (once by  $r$ , once by  $l$ ), and frequency updates are  $O(1)$  due to the fixed alphabet size (26 letters).
- **Space Complexity:**
  - $O(1)$  additional space. The frequency array uses constant space (size 26), independent of  $n$ .

The solution efficiently handles all cases within  $O(n)$  time and  $O(1)$  space, meeting the problem constraints.