Longest Substring with K Distinct Characters (medium)

We'll cover the following

- Problem Statement
- Try it yourself
- Solution
- Code
  - Time Complexity
  - Space Complexity

Problem Statement

Given a string, find the length of the **longest substring** in it **with no more than K distinct characters**.

Example 1:

```
Input: String="araaci", K=2
Output: 4
Explanation: The longest substring with no more than '2' distinct characters is "araa".
```

Example 2:

```
Input: String="araaci", K=1
Output: 2
Explanation: The longest substring with no more than '1' distinct characters is "aa".
```

Example 3:

```
Input: String="cbbebi", K=3
Output: 5
Explanation: The longest substrings with no more than '3' distinct characters are "cbbeb" & "bbe
bi".
```

Try it yourself

Try solving this question here:

Java Python3 JS C++

```
def longest_substring_with_k_distinct(str, k):
  # TODO: Write your code here
  return -1
```

Test Save Reset

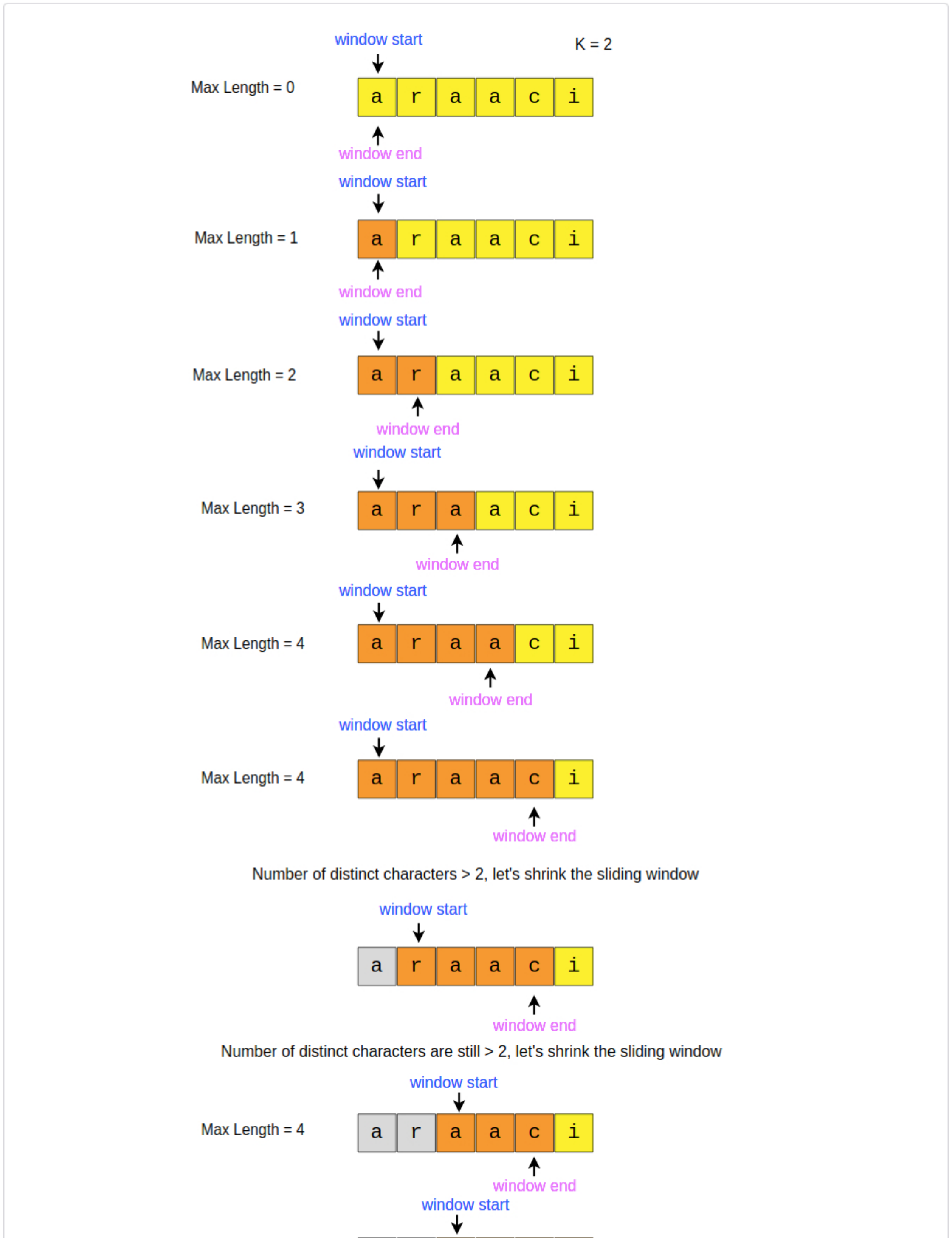
Solution

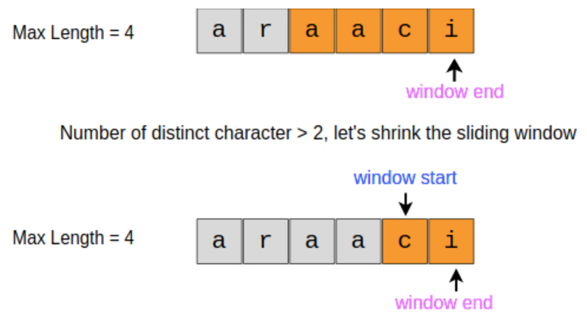
This problem follows the **Sliding Window** pattern, and we can use a similar dynamic sliding window strategy as discussed in Smallest Subarray with a given sum. We can use a **HashMap** to remember the frequency of each character we have processed. Here is how we will solve this problem:

1. First, we will insert characters from the beginning of the string until we have 'K' distinct characters in the **HashMap**.

2. These characters will constitute our sliding window. We are asked to find the longest such window having no more than ‘K’ distinct characters. We will remember the length of this window as the longest window so far.
3. After this, we will keep adding one character in the sliding window (i.e., slide the window ahead) in a stepwise fashion.
4. In each step, we will try to shrink the window from the beginning if the count of distinct characters in the **HashMap** is larger than ‘K.’ We will shrink the window until we have no more than ‘K’ distinct characters in the **HashMap**. This is needed as we intend to find the longest window.
5. While shrinking, we’ll decrement the character’s frequency going out of the window and remove it from the **HashMap** if its frequency becomes zero.
6. At the end of each step, we’ll check if the current window length is the longest so far, and if so, remember its length.

Here is the visual representation of this algorithm for the Example-1:





Code

Here is how our algorithm will look:

```python
def longest_substring_with_k_distinct(str1, k):
  window_start = 0
  max_length = 0
  char_frequency = {}

  # in the following loop we'll try to extend the range [window_start, window_end]
  for window_end in range(len(str1)):
    right_char = str1[window_end]
    if right_char not in char_frequency:
      char_frequency[right_char] = 0
    char_frequency[right_char] += 1

    # shrink the sliding window, until we are left with 'k' distinct characters in the char_frequency
    while len(char_frequency) > k:
      left_char = str1[window_start]
      char_frequency[left_char] -= 1
      if char_frequency[left_char] == 0:
        del char_frequency[left_char]
      window_start += 1  # shrink the window
    # remember the maximum length so far
    max_length = max(max_length, window_end-window_start + 1)
  return max_length


def main():
  print("Length of the longest substring: " + str(longest_substring_with_k_distinct("araaci", 2)))
  print("Length of the longest substring: " + str(longest_substring_with_k_distinct("araaci", 1)))
  print("Length of the longest substring: " + str(longest_substring_with_k_distinct("cbbebi", 3)))


main()
```

Time Complexity

The above algorithm's time complexity will be $O(N)$, where 'N' is the number of characters in the input string. The outer `for` loop runs for all characters, and the inner `while` loop processes each character only once; therefore, the time complexity of the algorithm will be $O(N + N)$, which is asymptotically equivalent to $O(N)$.

Space Complexity

The algorithm's space complexity is $O(K)$, as we will be storing a maximum of 'K+1' characters in the HashMap.

← Back
Smallest Subarray with a given sum (e...

Next →
Fruits into Baskets (medium)

✓ Completed

Report an Issue Ask a Question