**Longest Substring with At Most Two Distinct Characters**

Question

Given a string **s** , find the length of the longest substring **t**that contains **at most**2 distinct characters.

**Example 1:**

**Input:** "eceba"

**Output:** 3

**Explanation: t** is "ece" which its length is 3.

**Example 2:**

**Input:** "ccaabbb"

**Output:** 5

**Explanation: t** is "aabbb" which its length is 5.

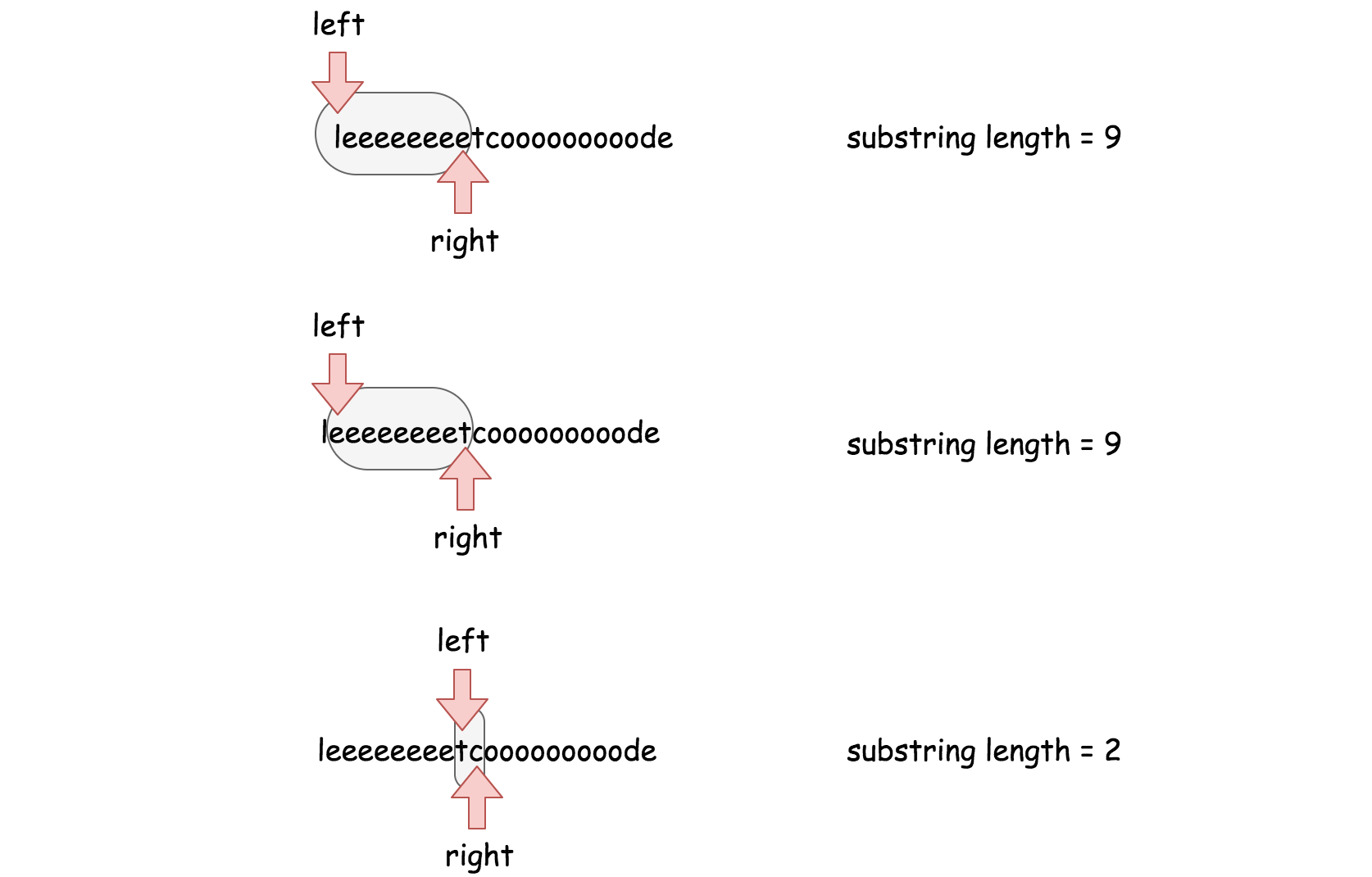
# **Solution**

#### **Approach 1: Sliding Window**

**Intuition**

To solve the problem in one pass let's use here sliding window approach with two set pointers left and right serving as the window boundaries.

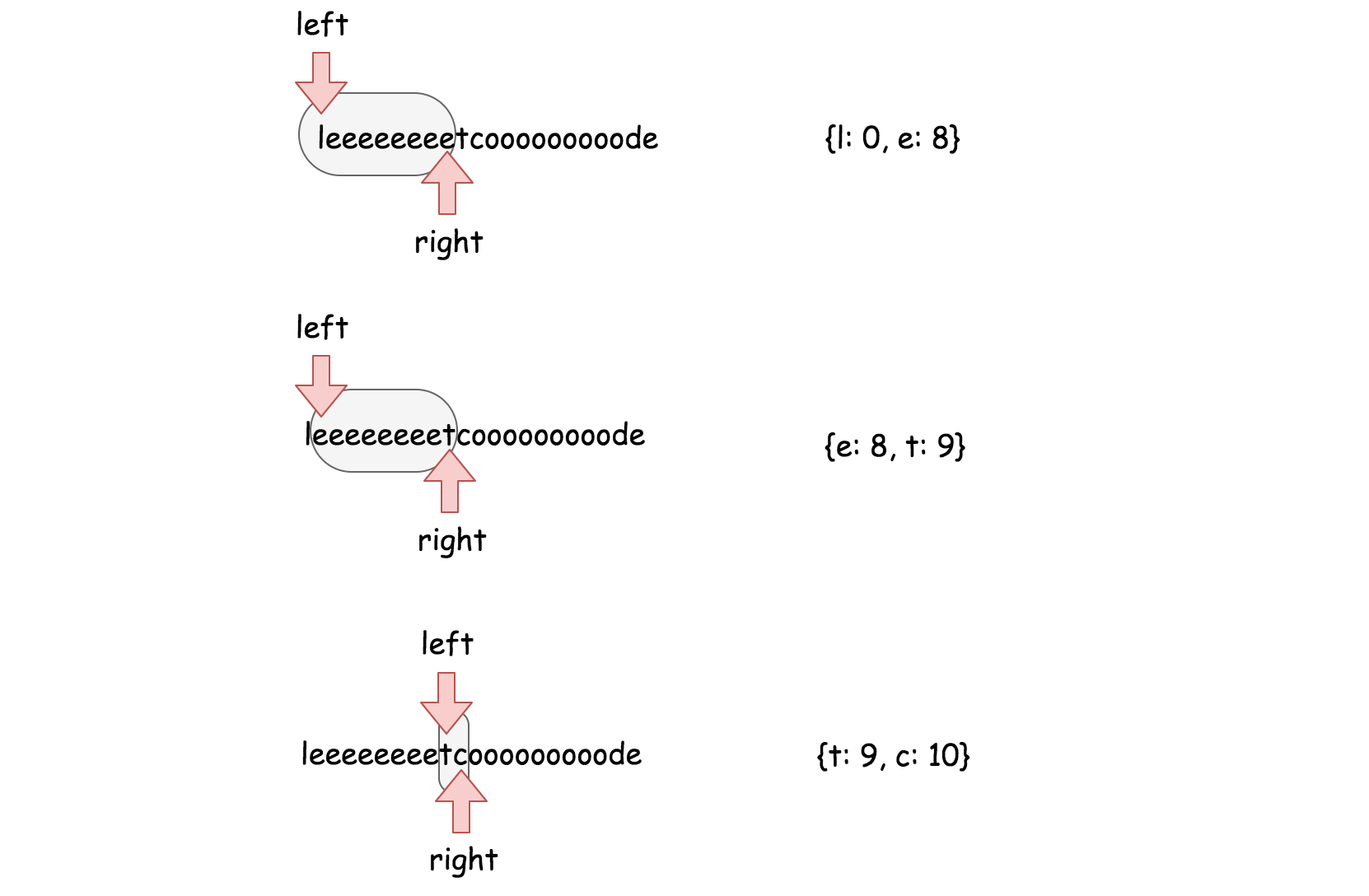
The idea is to set both pointers in the position 0 and then move right pointer to the right while the window contains not more than two distinct characters. If at some point we've got 3 distinct characters, let's move left pointer to keep not more than 2 distinct characters in the window.



Basically that's the algorithm : to move sliding window along the string, to keep not more than 2 distinct characters in the window, and to update max substring length at each step.

There is just one more question to reply - how to move the left pointer to keep only 2 distinct characters in the string?

Let's use for this purpose hashmap containing all characters in the sliding window as keys and their rightmost positions as values. At each moment, this hashmap could contain not more than 3 elements.



For example, using this hashmap one knows that the rightmost position of character e in "eeeeeeeet" window is 8 and so one has to move left pointer in the position 8 + 1 = 9 to exclude the character e from the sliding window.

Do we have here the best possible time complexity? Yes, we do - it's the only one pass along the string with N characters and the time complexity is \mathcal{O}(N)O(*N*).

**Algorithm**

Now one could write down the algortihm.

* Return N if the string length N is smaller than 3.
* Set both set pointers in the beginning of the string left = 0 and right = 0 and init max substring length max\_len = 2.
* While right pointer is less than N:
  + If hashmap contains less than 3 distinct characters, add the current character s[right] in the hashmap and move right pointer to the right.
  + If hashmap contains 3 distinct characters, remove the leftmost character from the hashmap and move the left pointer so that sliding window contains again 2 distinct characters only.
  + Update max\_len.

Coding Solution

Java

|  |
| --- |
| class Solution {  public int lengthOfLongestSubstringTwoDistinct(String s) {  int n = s.length();  if (n < 3) return n;  // sliding window left and right pointers  int left = 0;  int right = 0;  // hashmap character -> its rightmost position  // in the sliding window  HashMap<Character, Integer> hashmap = new HashMap<Character, Integer>();  int max\_len = 2;  while (right < n) {  // when the slidewindow contains less than 3 characters  hashmap.put(s.charAt(right), right++);  // slidewindow contains 3 characters  if (hashmap.size() == 3) {  // delete the leftmost character  int del\_idx = Collections.min(hashmap.values());  hashmap.remove(s.charAt(del\_idx));  // move left pointer of the slidewindow  left = del\_idx + 1;  }  max\_len = Math.max(max\_len, right - left);  }  return max\_len;  }  } |

Python

|  |
| --- |
| from collections import defaultdict  class Solution:  def lengthOfLongestSubstringTwoDistinct(self, s: 'str') -> 'int':  n = len(s)  if n < 3:  return n  # sliding window left and right pointers  left, right = 0, 0  # hashmap character -> its rightmost position  # in the sliding window  hashmap = defaultdict()  max\_len = 2  while right < n:  # when the slidewindow contains less than 3 characters  hashmap[s[right]] = right  right += 1  # slidewindow contains 3 characters  if len(hashmap) == 3:  # delete the leftmost character  del\_idx = min(hashmap.values())  del hashmap[s[del\_idx]]  # move left pointer of the slidewindow  left = del\_idx + 1  max\_len = max(max\_len, right - left)  return max\_len |

**Complexity Analysis**

* Time complexity : \mathcal{O}(N)O(*N*) where N is a number of characters in the input string.
* Space complexity : \mathcal{O}(1)O(1) since additional space is used only for a hashmap with at most 3 elements.

**Problem generalization**

The same sliding window approach could be used to solve the generalized problem :

[Longest Substring with At Most K Distinct Characters](https://leetcode.com/problems/longest-substring-with-at-most-k-distinct-characters/)