159. Longest Substring with at Most Two Distinct Characters

Feb. 13, 2019 | 37.7K views

Previous Next

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.
```

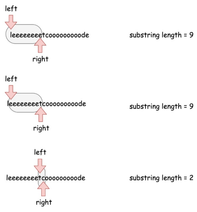
Solution

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.

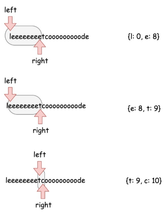
The idea is to set both pointers in the position `0` and then move `right` pointer to the right while the window contains not more than two distinct characters. If at some point we've got `3` distinct characters, let's move `left` pointer to keep not more than `2` distinct characters in the window.



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 $\mathcal{O}(N)$.

Algorithm

Now one could write down the algorithm.

- 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`.

Implementation

1 / 29

Java | Python

```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:
            # slidewindow contains less than 3 characters
            if len(hashmap) < 3:
                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
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N)$ where `N` is a number of characters in the input string.
- Space complexity : $\mathcal{O}(1)$ 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

Rate this article:

Previous Next

Comments: 21 Sort By

Type comment here... (Markdown is supported)

Preview Post

sanyam3 ★ 97 June 16, 2019 10:03 AM
Why is this problem classified as hard?
36 Share Reply
SHOW 5 REPLIES

rahulkun ★ 446 February 14, 2019 7:06 AM
less branchy approach would be put strlight() unconditionally and shirking the windows by increment left while size>2 O(N) time and O(2) space for hashmap

```
int lengthOfLongestSubstringTwoDistinct(string str) {
    int left = 0, right = 0;
    int[] hashmap = new int[128];
    int max_len = 2, distinct_count = 0;
    int[] to_add = new int[3];
    int tmp;
    while (right < str.length()) {
        if (distinct_count < 3) {
            tmp = str.charAt(right);
            to_add[distinct_count] = tmp;
            if (hashmap[tmp] == 0) distinct_count++;
            hashmap[tmp]++;
            right++;
        }
        else {
            tmp = str.charAt(left);
            hashmap[tmp]--;
            if (hashmap[tmp] == 0)
                distinct_count--;
            left++;
        }
        if (right - left > max_len)
            max_len = right - left;
    }
    return max_len;
}
```

Read More

6 Share Reply
SHOW 1 REPLY

logansu ★ 30 May 27, 2019 11:52 AM
The conditional `if len(hashmap) < 3:` is unnecessary since the next conditional ensures the count will always be at most 2.
8 Share Reply

cjim8889 ★ 20 February 16, 2019 4:06 AM
i know which company uses this question , lol
3 Share Reply
SHOW 2 REPLIES

saurabhchris1 ★ 12 January 8, 2020 4:16 AM
Python Time O(n) and Space O(1) no hashmap

```
class Solution:
    def lengthOfLongestSubstringTwoDistinct(self, s: str) -> int:
        left, right = 0, 0
        hashmap = defaultdict()
        max_len = 2

        while right < len(s):
            if len(hashmap) < 3:
                hashmap[s[right]] = right
                right += 1
            if len(hashmap) == 3:
                del_idx = min(hashmap.values())
                del hashmap[s[del_idx]]
                left = del_idx + 1
            if right - left > max_len:
                max_len = right - left
        return max_len
```

Read More

1 Share Reply

yaokzhan ★ 0 March 15, 2019 1:20 AM
11 lines java array solution:

```
public int lengthOfLongestSubstringTwoDistinct(String s) {
    int[] map = new int[128];
    int count = 0, l = 0, r = 0, tmp, max = 0;
    while (r < s.length()) {
        tmp = s.charAt(r);
        if (map[tmp] == 0) count++;
        map[tmp]++;
        r++;
        while (count > 2) {
            tmp = s.charAt(l);
            map[tmp]--;
            if (map[tmp] == 0)
                count--;
            l++;
        }
        if (r - l > max)
            max = r - l;
    }
    return max;
}
```

Read More

0 Share Reply
SHOW 1 REPLY

aliabd11 ★ 0 May 19, 2019 7:22 AM
Isn't determining the length of the hashmap, O(n)? So the time complexity of this solution would not be O(n).
0 Share Reply
SHOW 2 REPLIES

mabunday ★ 1 February 17, 2019 2:56 AM
Better Python 3 solution: https://leetcode.com/problems/longest-substring-with-at-most-k-distinct-characters/discuss/238313/python-3-true-on-runtime-ok-space-using-ordereddict
0 Share Reply
SHOW 2 REPLIES

user3662 ★ 0 February 16, 2019 4:43 PM
using only one array solution:

```
public static int kUnique(String str, int k){
        k+2; // remove k+2 for k unique characters
        int[] count = new int[256];
        int[] lastPos = new int[256];
        int start = 0, end = 0, maxLen = 0, distinct = 0;
        for (int i = 0; i < str.length(); i++) {
            int c = str.charAt(i);
            if (count[c] == 0) distinct++;
            count[c]++;
            lastPos[c] = i;
            if (distinct > k) {
                int minPos = Integer.MAX_VALUE;
                for (int j = 0; j < 256; j++) {
                    if (count[j] > 0 && lastPos[j] < minPos)
                        minPos = lastPos[j];
                }
                count[str.charAt(minPos)] = 0;
                distinct--;
                start = minPos + 1;
            }
            if (end - start > maxLen)
                maxLen = end - start;
            end++;
        }
        return maxLen;
}
```

Read More

0 Share Reply
SHOW 2 REPLIES

khotiaintseva ★ 0 February 16, 2019 12:22 AM
Why does the python solution use defaultdict instead of a regular dict?
0 Share Reply
SHOW 1 REPLY

1 2 3