49. Group Anagrams

Medium 9522 318 Add to List Share

Given an array of strings `strs`, group **the anagrams** together. You can return the answer in **any order**.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:

```
Input: strs = ["eat","tea","tan","ate","nat","bat"]
Output: [["bat"],["nat","tan"],["ate","eat","tea"]]
```

Example 2:

```
Input: strs = [""]
Output: [[""]]
```

Example 3:

```
Input: strs = ["a"]
Output: [["a"]]
```

Constraints:

- `1 <= strs.length <= 10^4`
- `0 <= strs[i].length <= 100`
- `strs[i]` consists of lowercase English letters.

Input: strs = ["eat","tea","tan","ate","nat","bat"]

After sort each anagram returns the same key.

Anagrams —>
[
[“eat”, “tea”, “ate”],
[“nat”, “tan”],
[“bat”]
]

key: aet —-> value:[“eat”, “tea”, “ate”],

Key: ant -> value: [“nat”, “tan”],

Key: abt -> value: [“bat”],

Map<String, List<String> >

Algorithm : O(n*klogk)

1. Iterate string arr.
2. Sort each String : O(klogk)
3. Upserts key:[sortedString] value:[currentIteratedString] in the map

Time Complexity: O(n*klogk)
Space Complexity : O(n)

By Considering space occupied for output.

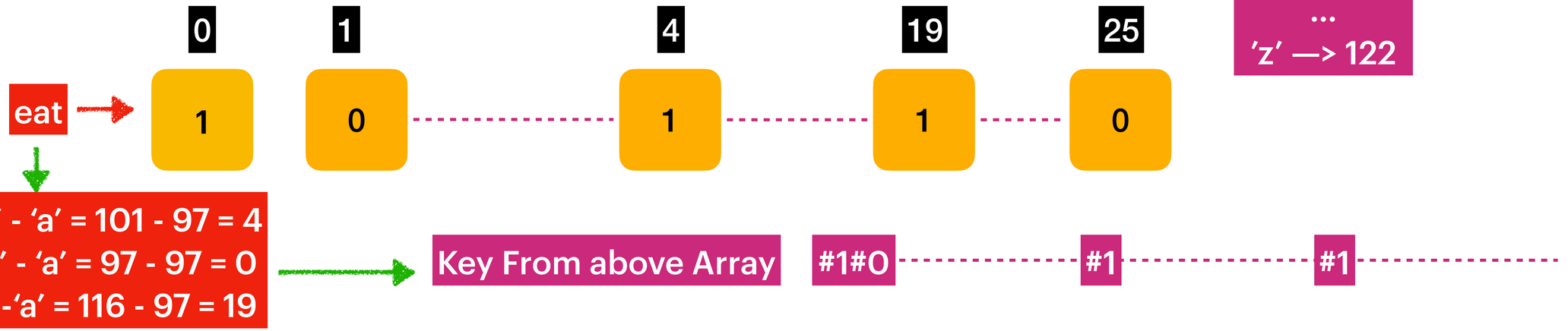
Input: strs = [“eat”,”tea","bat"]

[
[“eat”,”tea”],
[“bat”]
]

As per the constraints, input contains only lowercase letters.
So total max lower case char count is 26.
With this clue
we can generate hashKey without sorting.

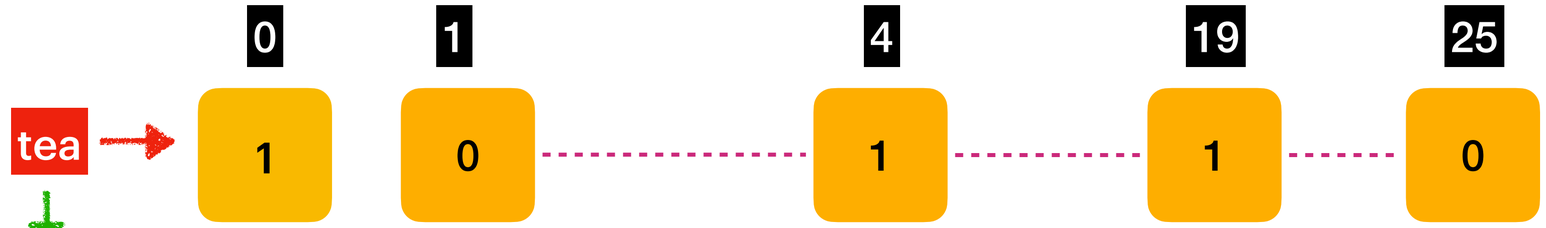
‘a’ -> 97
‘b’ -> 98
‘c’ -> 99
‘d’ -> 100
‘e’-> 101
....
‘t’->116
...
‘z’ —> 122

int[] arr = new int[26]



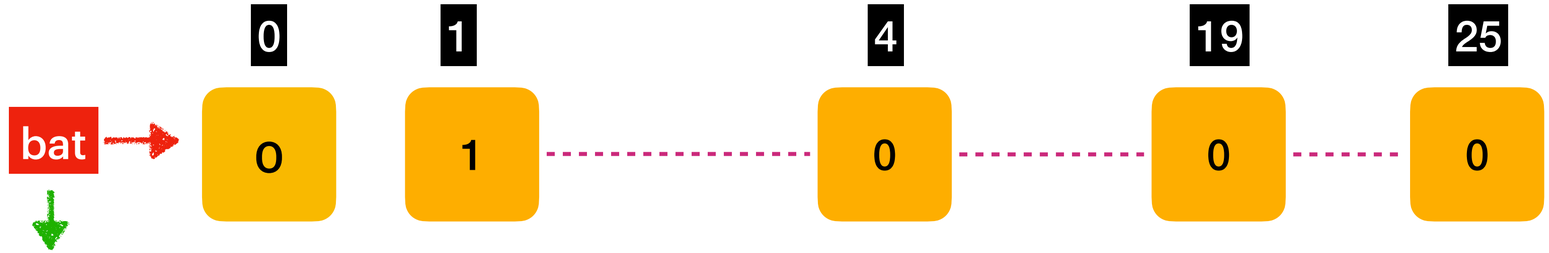
‘e’ -> ‘e’ - ‘a’ = 101 - 97 = 4
‘a’ -> ‘a’ - ‘a’ = 97 - 97 = 0
‘t’-> ‘t’ -‘a’ = 116 - 97 = 19

Key From above Array #1#0 #1 #1



‘t’-> ‘t’ -‘a’ = 116 - 97 = 19
‘e’ -> ‘e’ - ‘a’ = 101 - 97 = 4
‘a’ -> ‘a’ - ‘a’ = 97 - 97 = 0

Key From above Array #1#0 #1 #1



'b'-> 'b' -'a' = 98 - 97 = 1
'a' -> 'a' - 'a' = 97 - 97 = 0
't'-> 't' -'a' = 116 - 97 = 19

Key From above Array #0#1 #1 #1

Time Complexity: O(nk)
Space Complexity : O(n)

By Considering space occupied for output.

3. Longest Substring Without Repeating Characters

Medium 24110 1069 Add to List Share

Given a string `s`, find the length of the **longest substring** without repeating characters.

Example 1:

```
Input: s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
```

Example 2:

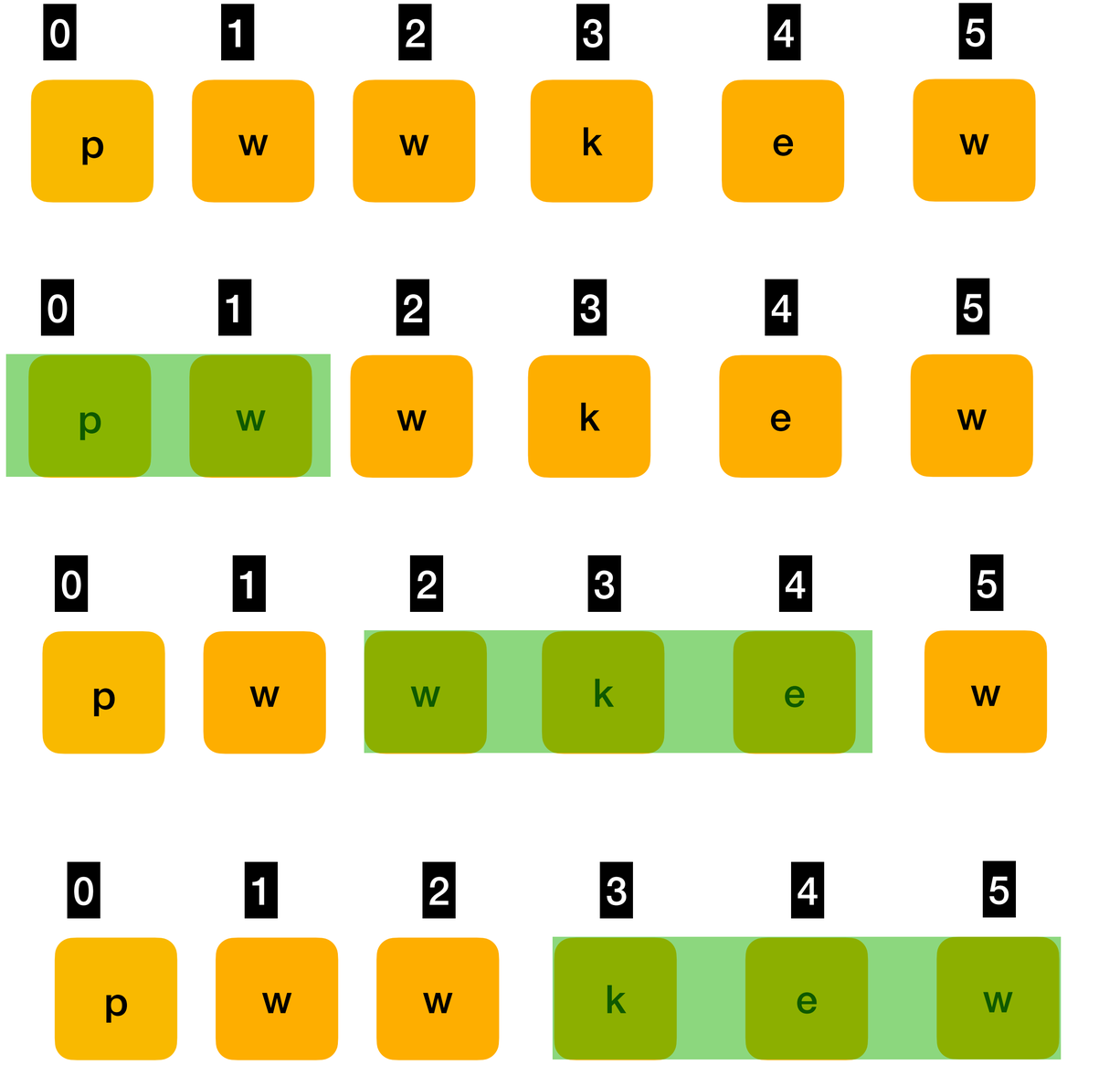
```
Input: s = "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
```

Example 3:

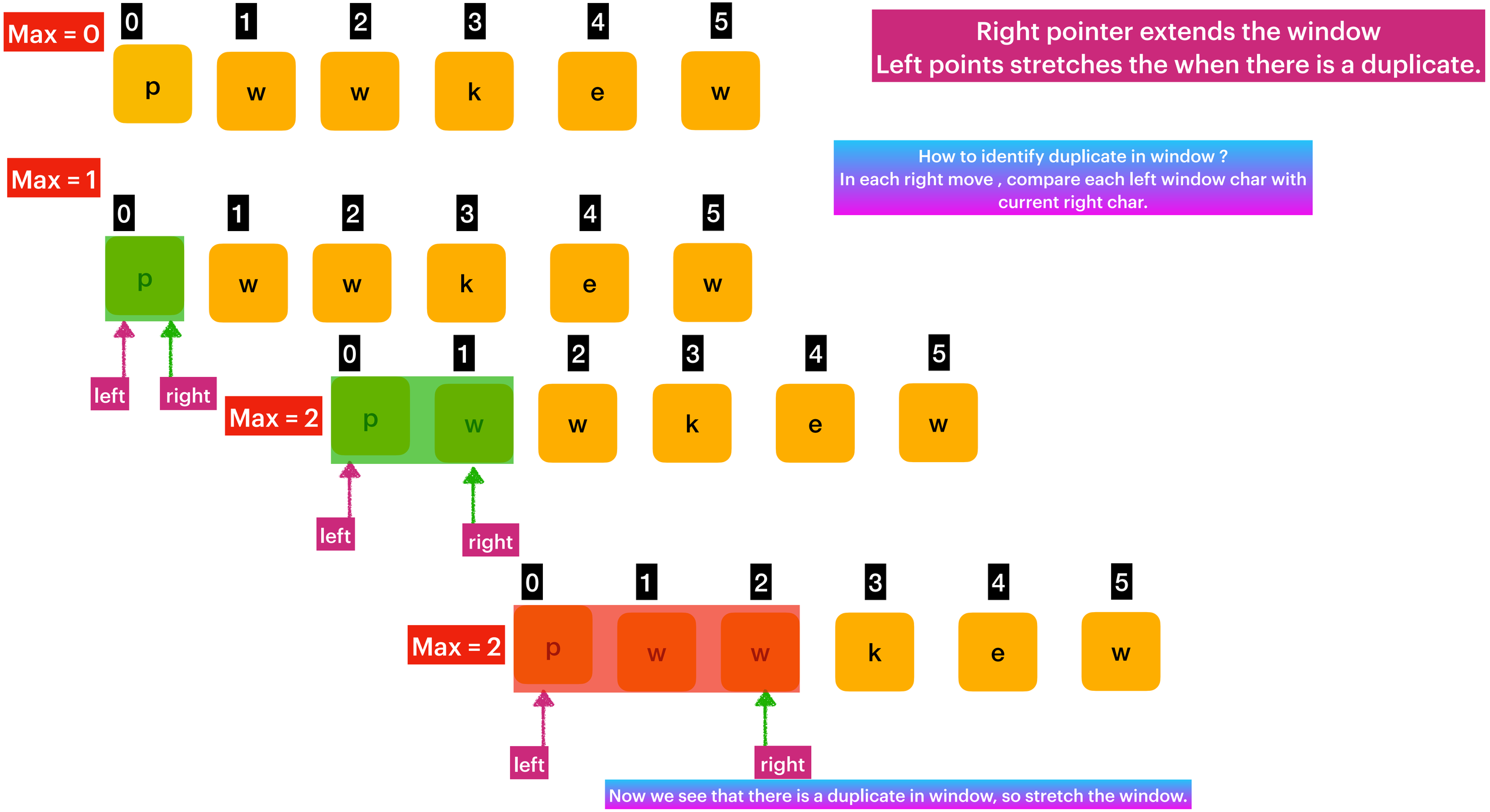
```
Input: s = "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Notice that the answer must be a substring, "pwke" is a
subsequence and not a substring.
```

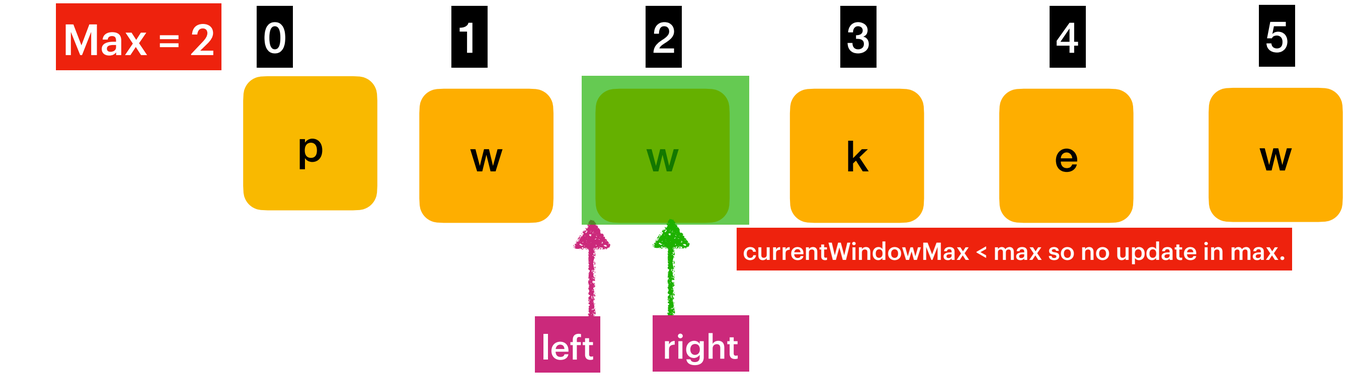
Constraints:

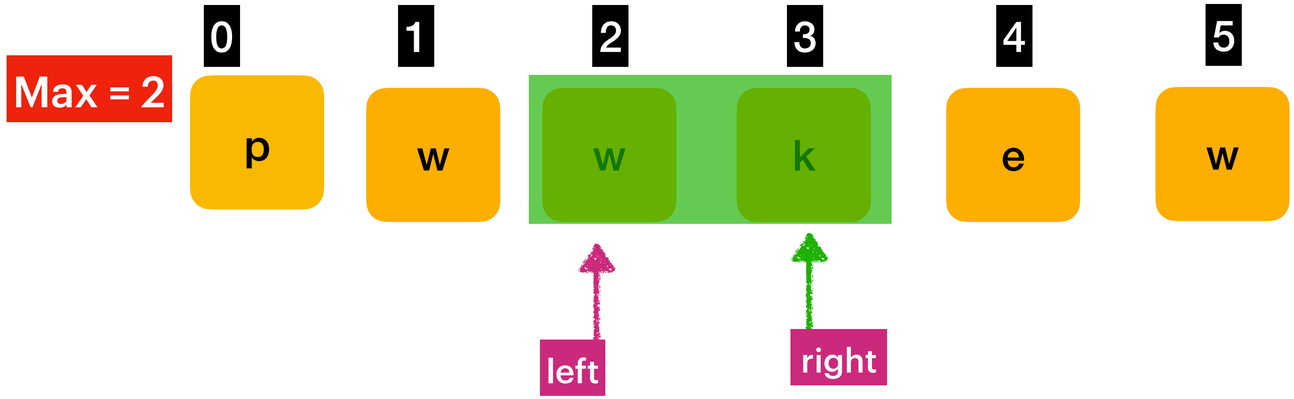
- `0 <= s.length <= 5 * 10^4`
- `s` consists of English letters, digits, symbols and spaces.

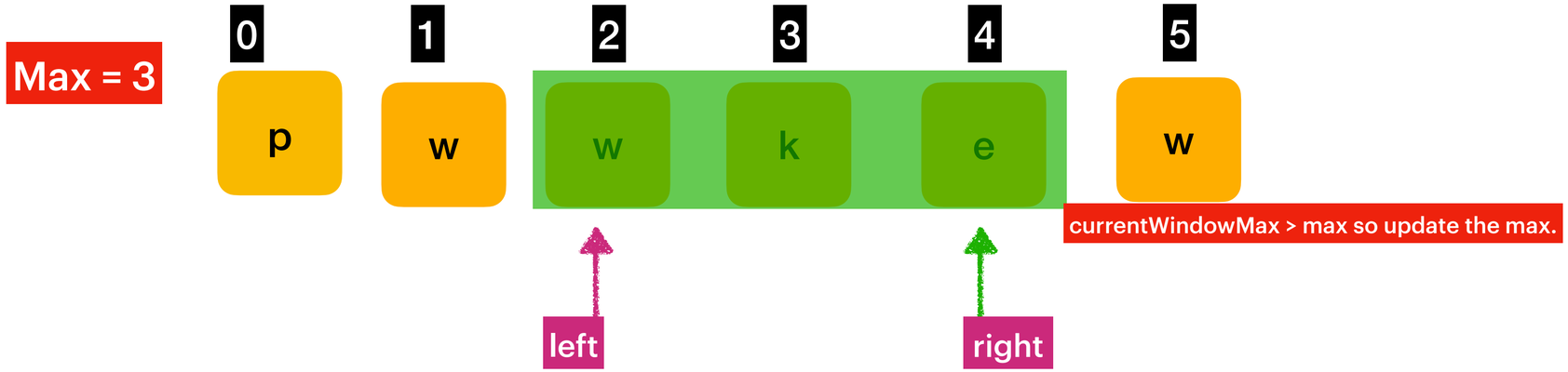


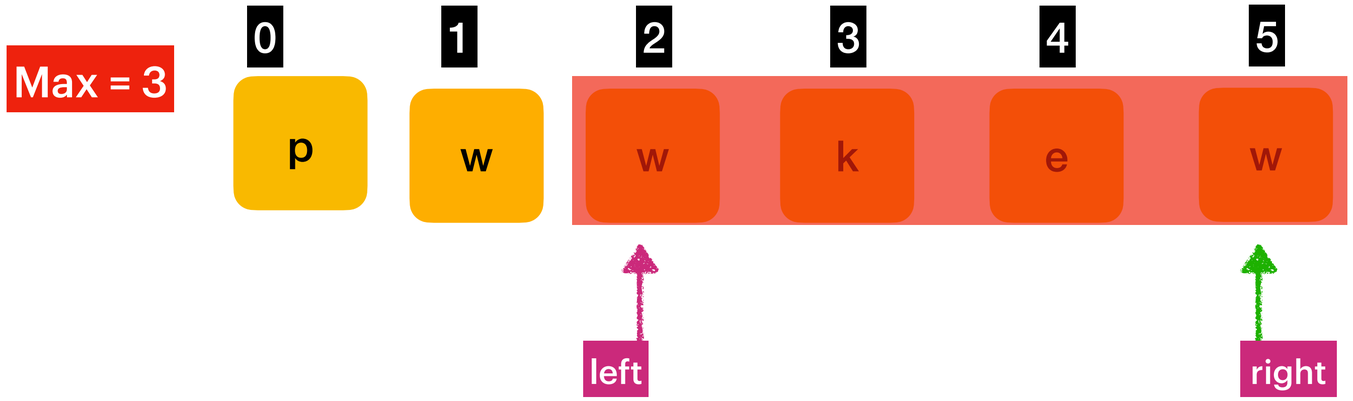
Expected Longest SubString without repeating characters :
either “wke” or “kew”
Return length = 3

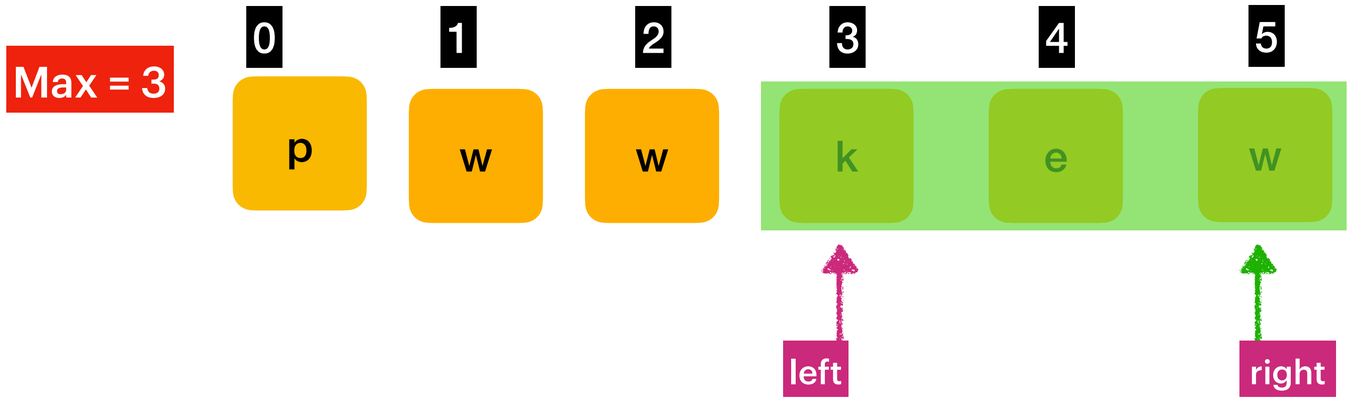

Max = 0
0
1
2
3
4
5
p
w
w
k
e
w
Right pointer extends the window
Left points stretches the when there is a duplicate.
How to identify duplicate in window ?
In each right move , compare each left window char with current right char.
Max = 1
left
right
Max = 2
left
right
Max = 2
left
right
Now we see that there is a duplicate in window, so stretch the window.


Max = 2
0
1
2
3
4
5
p
w
w
k
e
w
currentWindowMax < max so no update in max.
left
right


Max = 2
0
1
2
3
4
5
p
w
w
k
e
w
left
right


Max = 3
0
1
2
3
4
5
p
w
w
k
e
w
currentWindowMax > max so update the max.
left
right

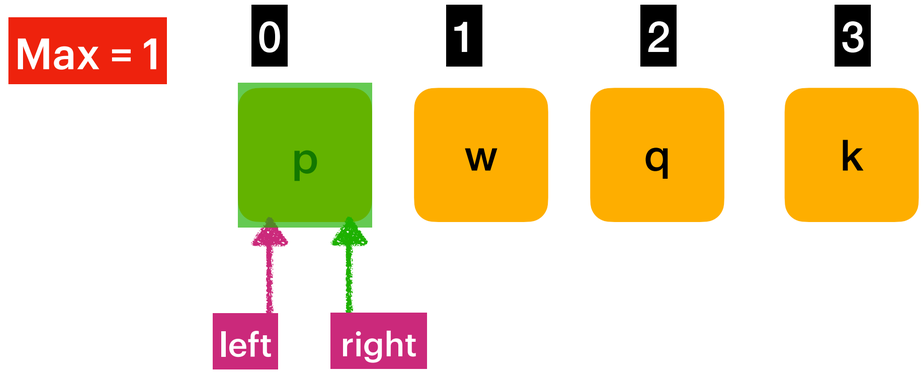




Return max : 3

Time Complexity : O(n^2)
[If all the characters are unique left will not move then
Leads to n^2 comparisons]

Space Complexity : O(1)



p -> visited —> 4 times
w -> visited —> 3 times
q -> visited —> 2 times
k —> 1 tme
-> n+(n-3)+(n-2)+..+1
= n(n-1)/2 = O(n^2)