Group Shifted Strings

can shift a string by shifting each of its letters to its successive letter.

For example, "abc" can be shifted to be "bcd".
We can keep shifting the string to form a sequence.

For example, we can keep shifting "abc" to form the sequence: "abc" -> "bcd" -> ... -> "xyz".
Given an array of strings strings, group all strings[i] that belong to the same shifting sequence.
You may return the answer in any order.

Constraints:
1 <= strings.length <= 200
1 <= strings[i].length <= 50
`strings[i]` consists of lowercase English letters.

Input: strings = ["abc","bcd","acef","xyz","az","ba","a","z"]
Output: [["acef"],["a","z"],["abc","bcd","xyz"],["az","ba"]]

Input: strings = ["a"]
Output: [[“a"]]

Lets look at Group Shifted Strings

a -> b -> c -> .....-> z -> a -> b

ab -> bc -> cd ->....->yz->za->ab

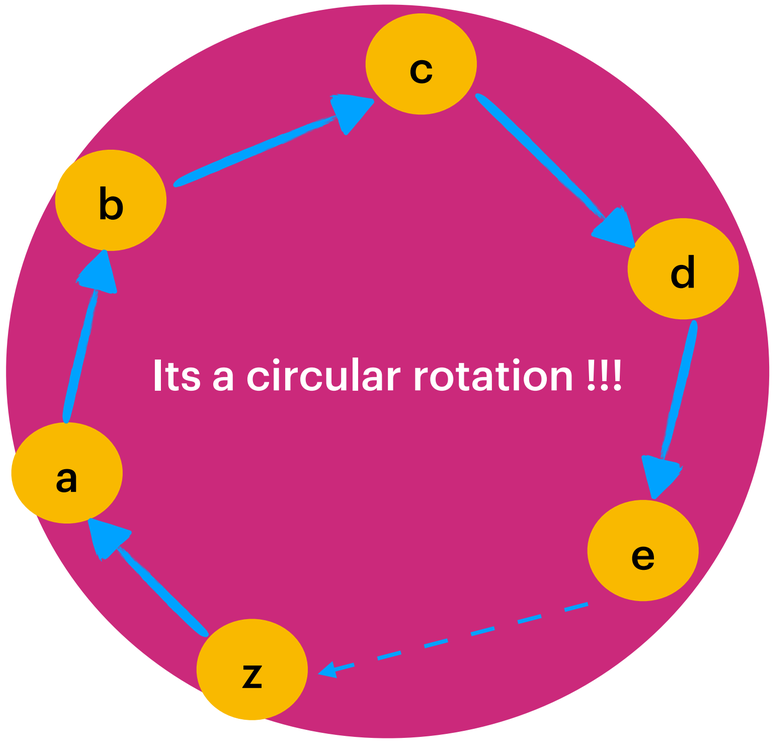
abc -> bcd -> cde -> ....> xyz -> yza ->ząb -> abc

az -> ba

We find distance of each character

It should be starting character of the string because rotation start from here !! This enables us to give GroupKey.

(s.charAt(i) - s.charAt(0) + 26) % 26



“abc”, ”bcd”, "xyz"

Distance between characters (c->b->a) is equal

“az”, ”ba”

Distance between characters (a -> b), (z->a) is equal

Let’s Apply math here to form a Group Key

Principle = (s.charAt(i) - startingChar + 26) % 26

“abc”, ”bcd”, "xyz"

Three Strings forms same key : 0|1|2|

String s = “abc”
startingChar = ‘a’
(a - a + 26) % 26 = 0
(b - a + 26) % 26 = (97-96+26)% 26 = 1
(c - a + 26) % 26 = (98-96+26)% 26 = 2

Finally “bcd” key = “0|1|2|”

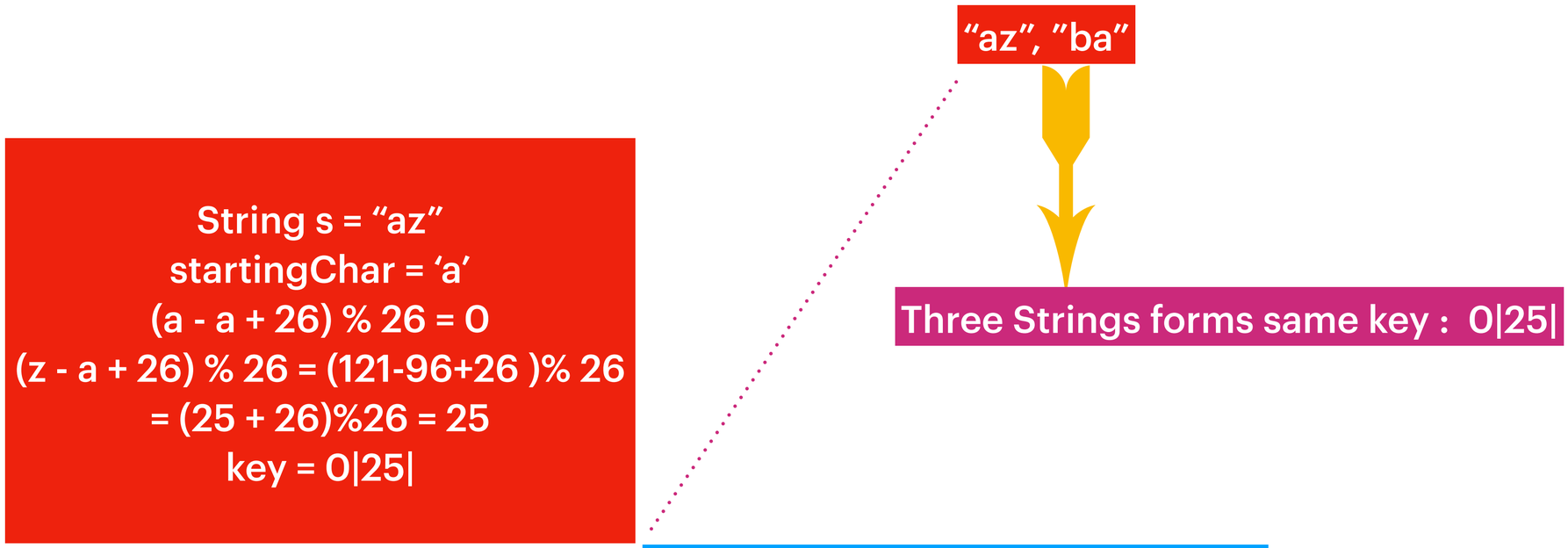
String s = “bcd”
startingChar = ‘b’
(b - b + 26) % 26 = 0
(c - b + 26) % 26 = (98-97+26)% 26 = 1
(d - b + 26) % 26 = (99-97+26)% 26 = 2

Finally “bcd” key = “0|1|2|”

String s = “cde”
startingChar = ‘c’
(c - c+ 26) % 26 = 0
(d - c + 26) % 26 = (99-98+26)% 26 = 1
(e - c + 26) % 26 = (100-98+26)% 26 = 2

Finally “cde” key = “0|1|2|”

Principle = (s.charAt(i) - startingChar + 26) % 26



Longest Substring Without Repeating Characters

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

Constraints:

$0 <= s.length <= 5 * 10_4$

s consists of English letters, digits, symbols and spaces.

Example 1:

Input: s = "abcabcbb"

Output: 3

Explanation: The answer is "abc", with the length of 3.

Example 2:

Input: s = “srinu nampalli”

Output: 6

Explanation: The answer is “srinu ", 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.

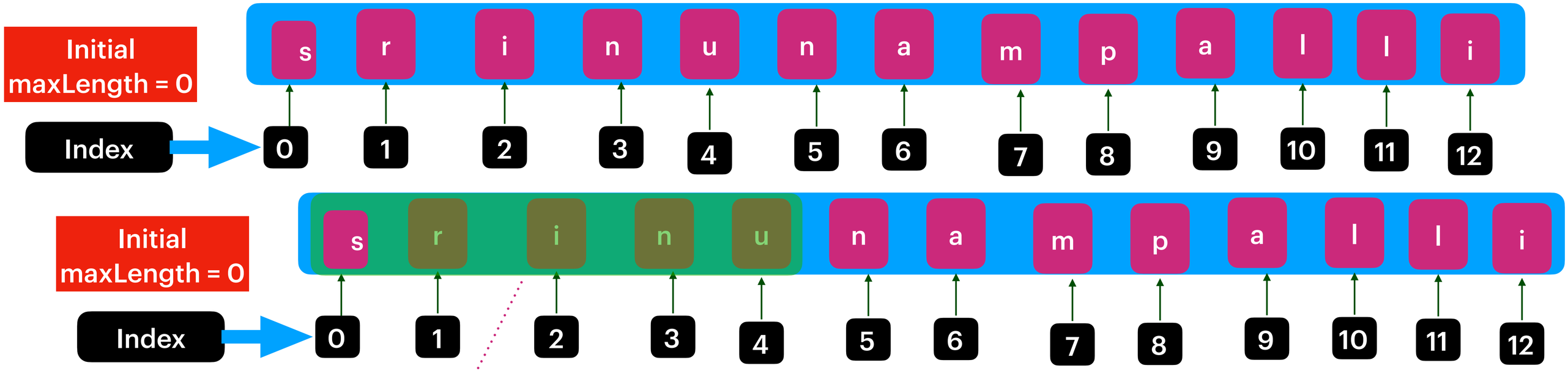
Example 4:

Input: s = ""

Output: 0

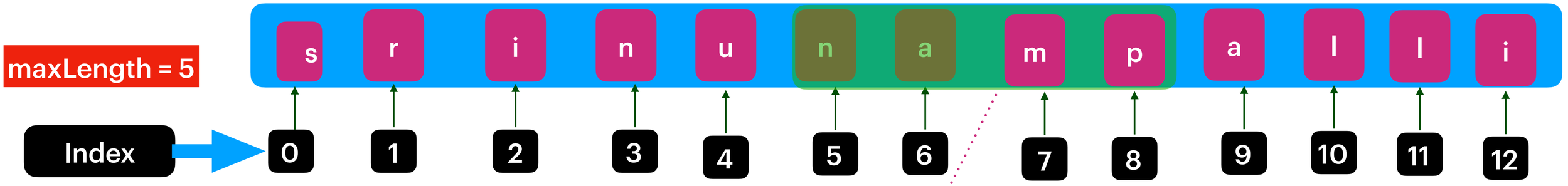
Abstract Solution with HashSet :

Take the character set, store the data into set
till the character is not repeated.
Update the maxLength and reset the size,
repeat the process from next character onwards .

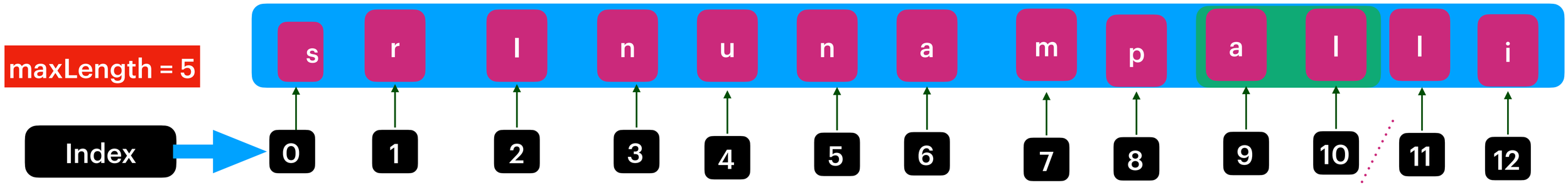


Store the data into Set<Character> till the character is not repeated :
set Size =5
current maxLength = 0 so update maxLength = 5
empty the Set.

Abstract Solution with HashSet :



Move on to next Character, Store the data into Set<Character> till the character is not repeated :
set Size =4
current maxLength = 5 so no update maxLength = 5
empty the Set.

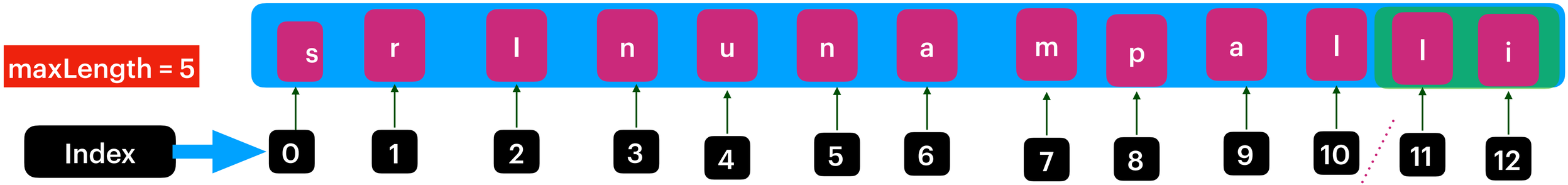


Move on to next Character, Store the data into Set<Character> till the character is not repeated :
set Size = 2
current maxLength = 5 so no update on maxLength = 5
empty the Set.

Reached end of the array, Return Max Length 6 :

Is our algorithm is perfect ?

Abstract Solution with HashSet :



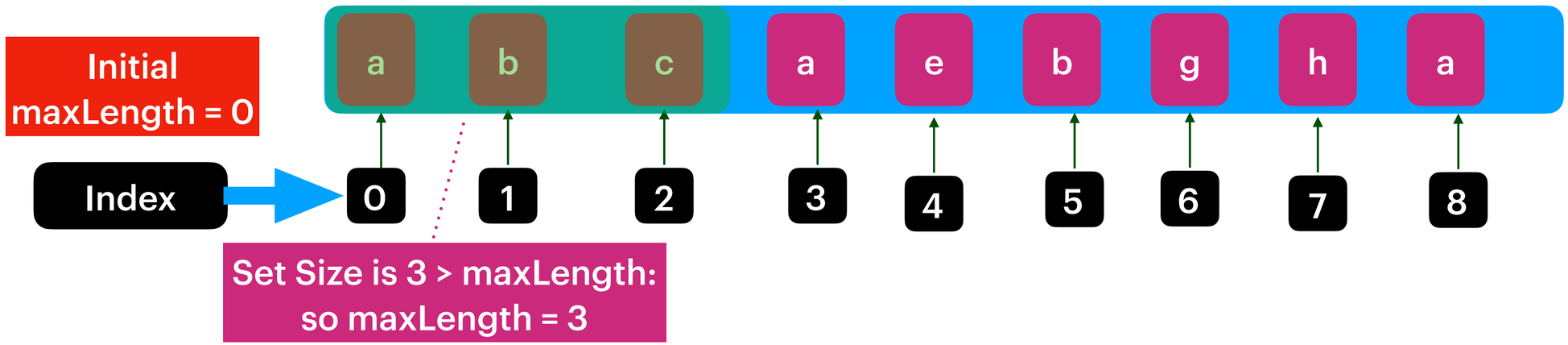
Move on to next Character, Store the data into Set<Character> till the character is not repeated :
set Size = 2
current maxLength = 5 so no update on maxLength = 5
empty the Set.

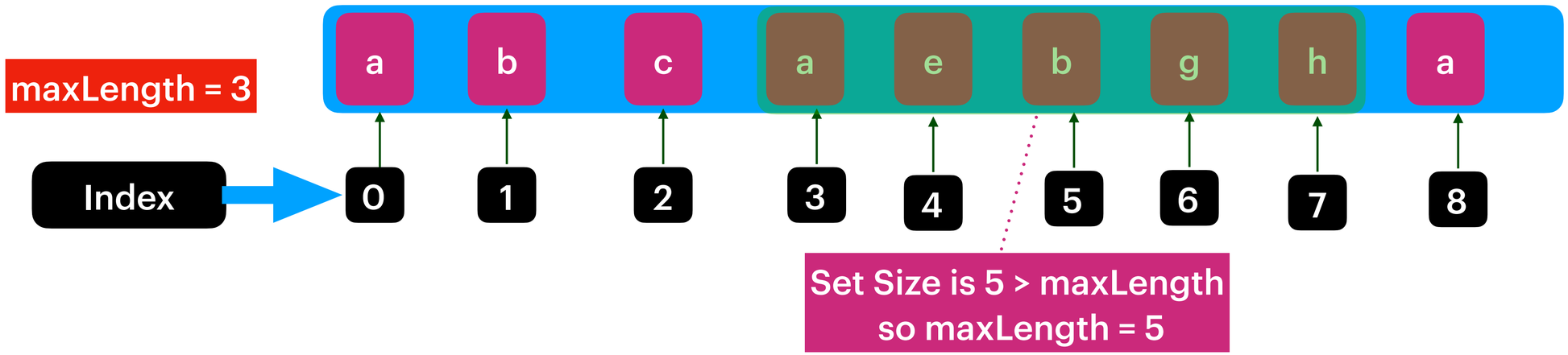
Reached end of the array, Return Max Length 5 :

Is our algorithm is perfect ?

Abstract Solution with HashSet :

Is our algorithm is perfect ?
What if our the SubString length is higher if we consider character before the next repeated value !!!!

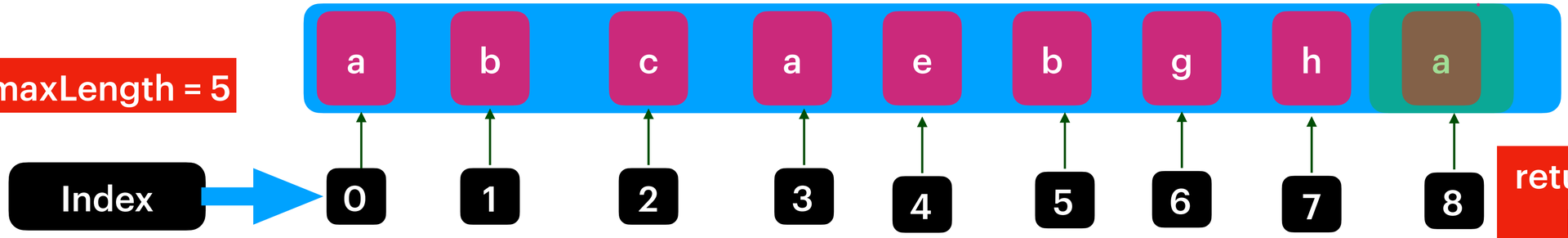




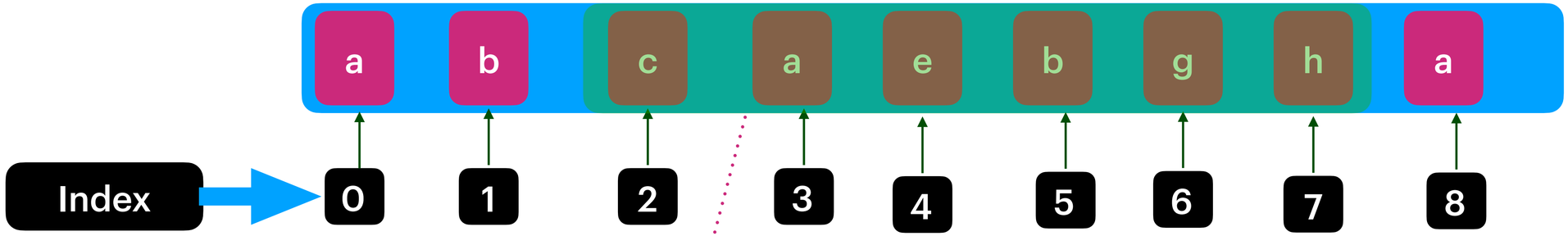
Abstract Solution with HashSet :

Set Size is 1: < maxLength
so no update i.e maxLength = 5

maxLength = 5



return maxLength = 5.
Is this currect ?

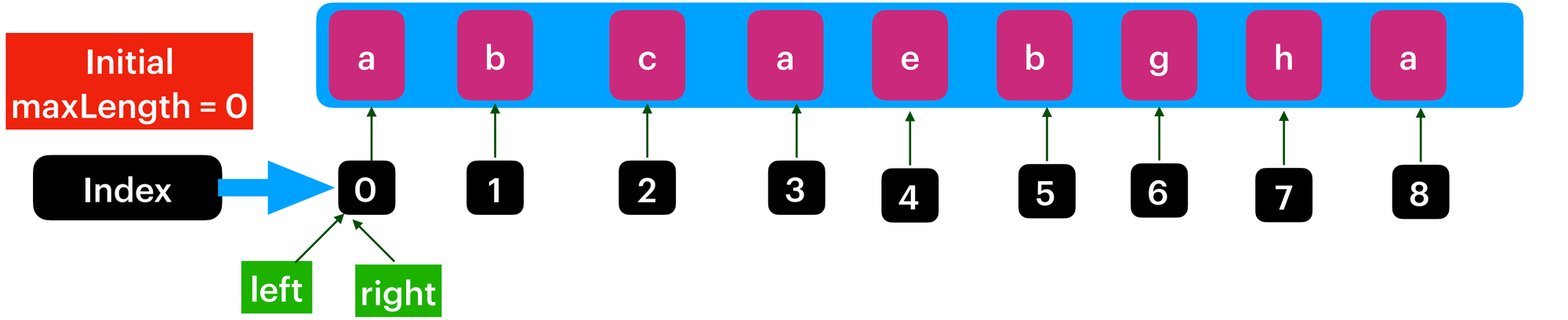


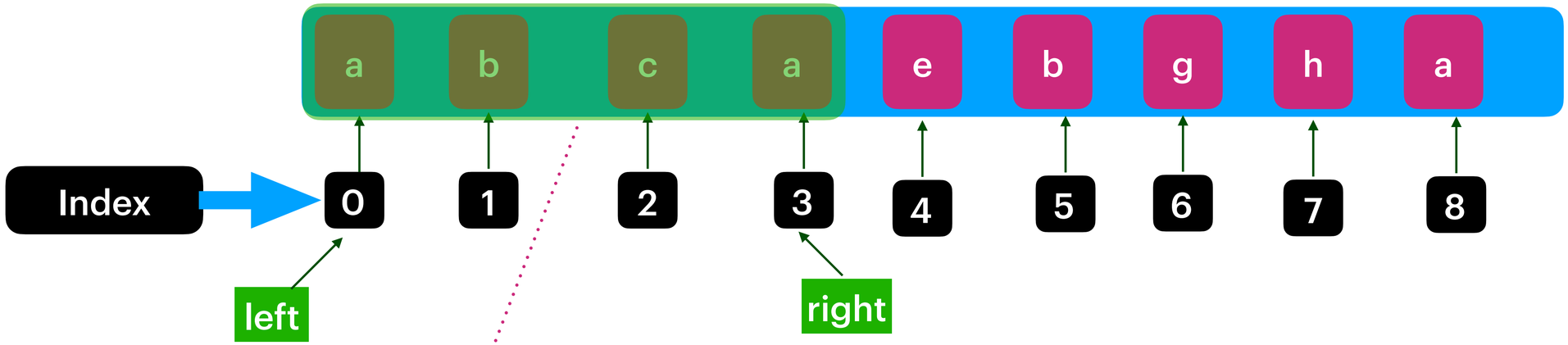
Expected Max Length is : 6

Mistake from this algorithm is we are tracking left part of the window !!!!

Abstract Solution2 : Maintaining left and right pointer !!!

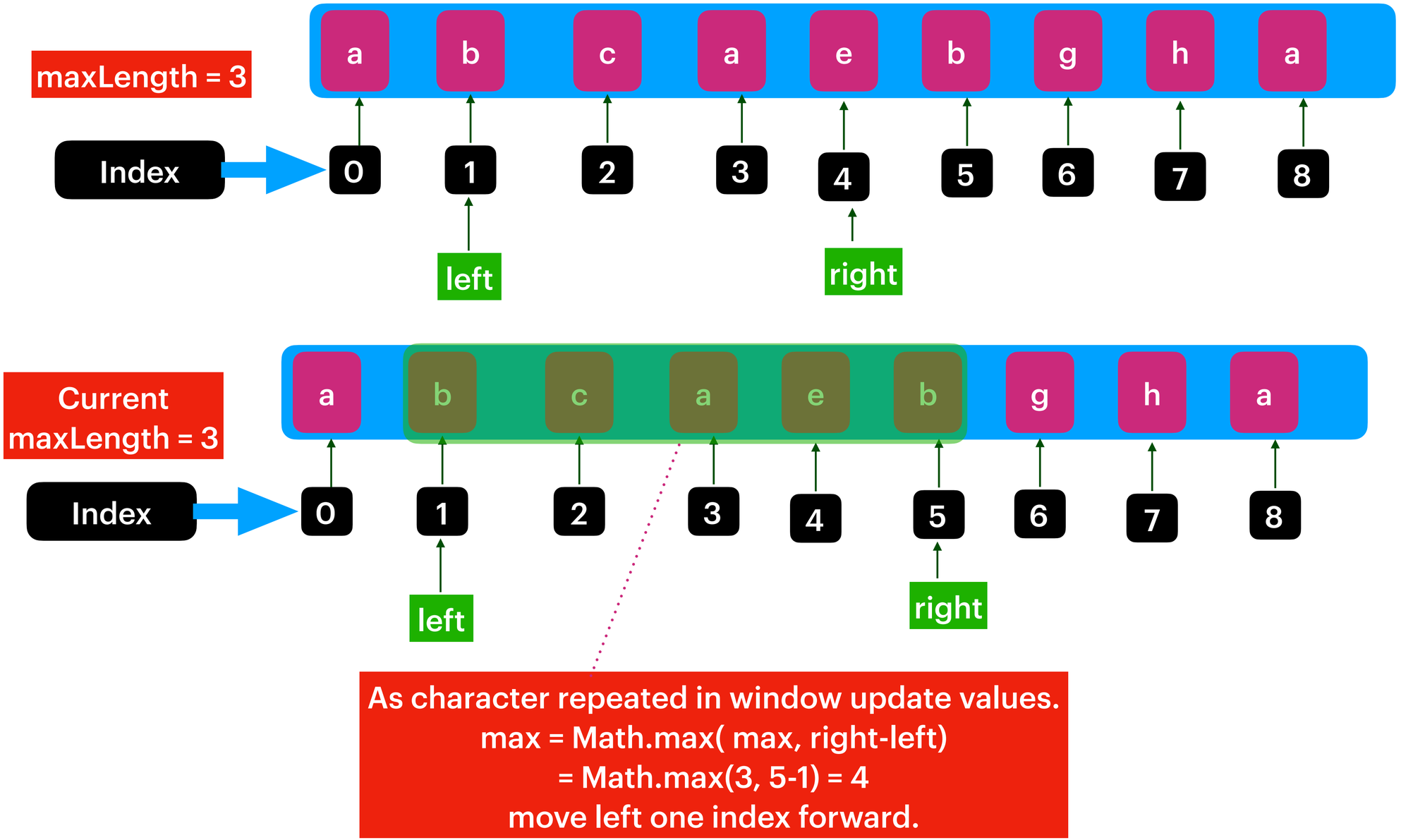
Move the right pointer till the end,
Left pointer will be moved only when the character is repeated.
At each window take the diff of right & left pointer.



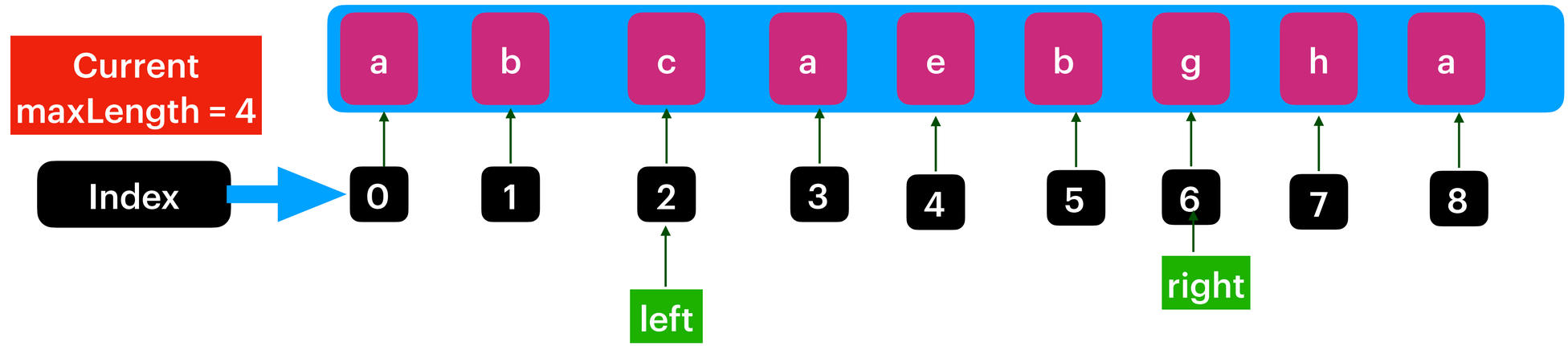


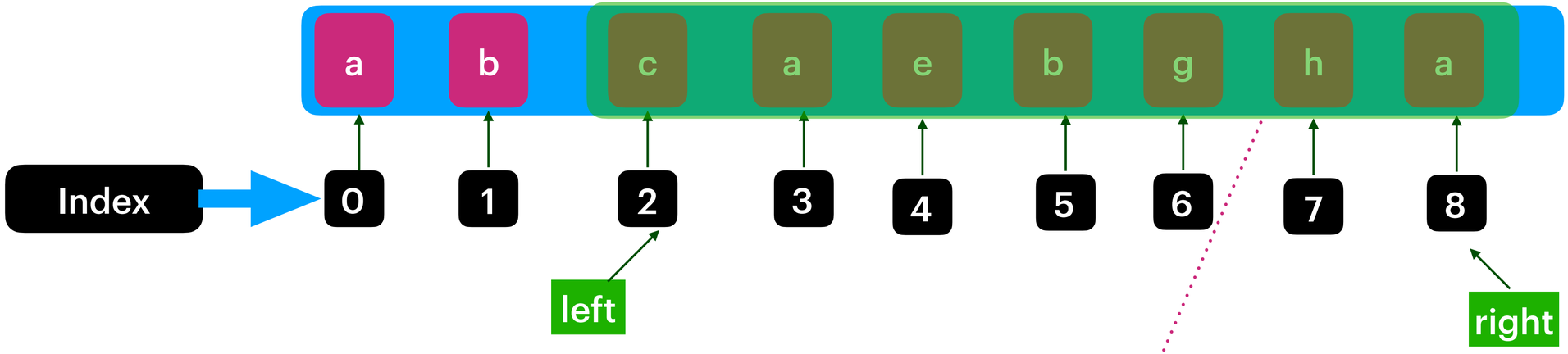
As character repeated in window update values.
max = Math.max(max, right-left)
= Math.max(0, 3-0) = 3
move left pointer to one index forward.

Abstract Solution2 : Maintaining left and right pointer !!!



Abstract Solution2 : Maintaining left and right pointer !!!

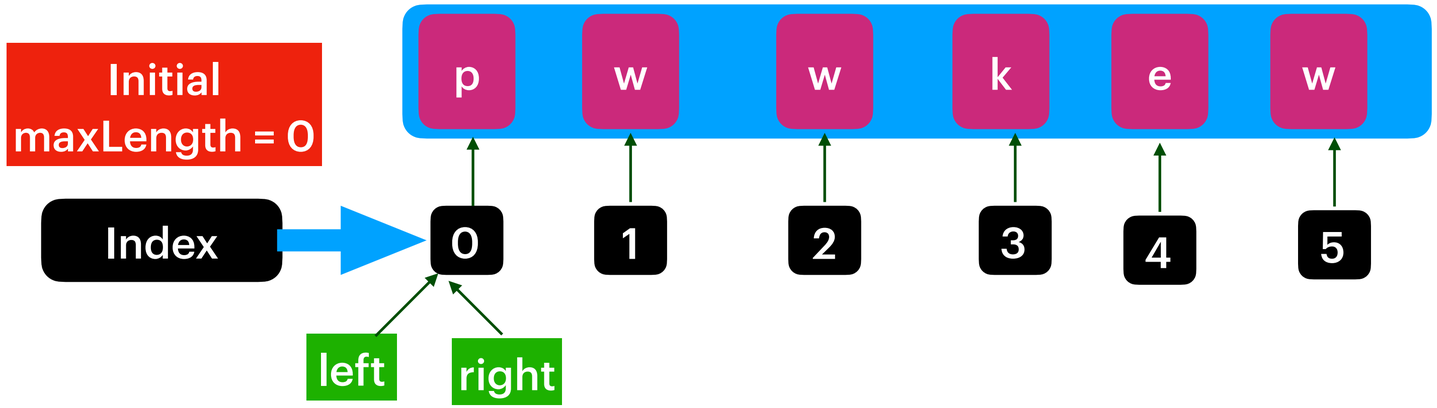


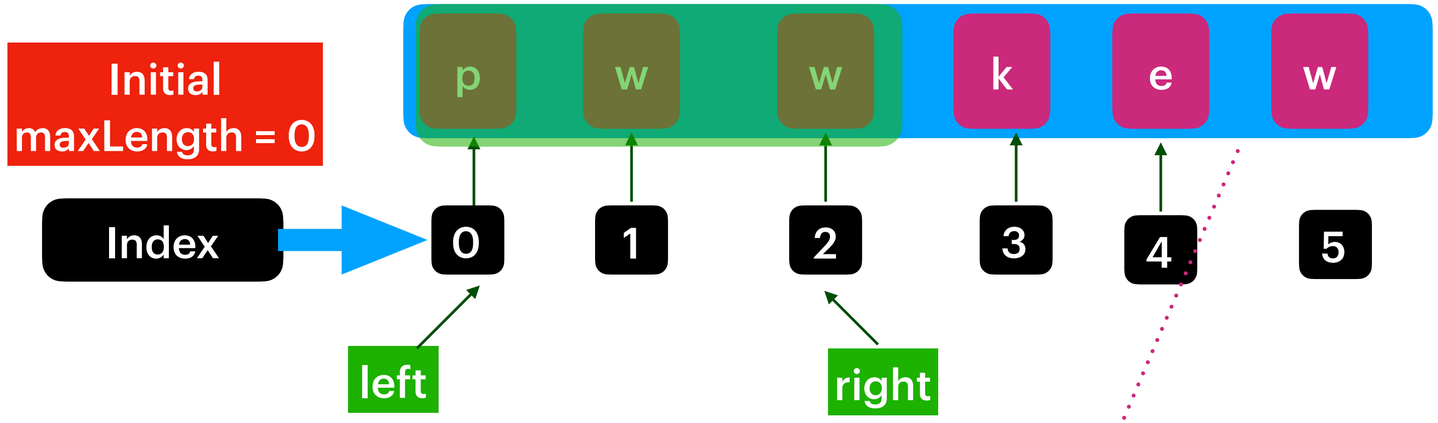


As character repeated in window update values.
max = Math.max(max, right-left)
= Math.max(4, 8-2) = 6
move left one index forward.

Is this Solution correct ? What if left side of window characters are duplicate ?

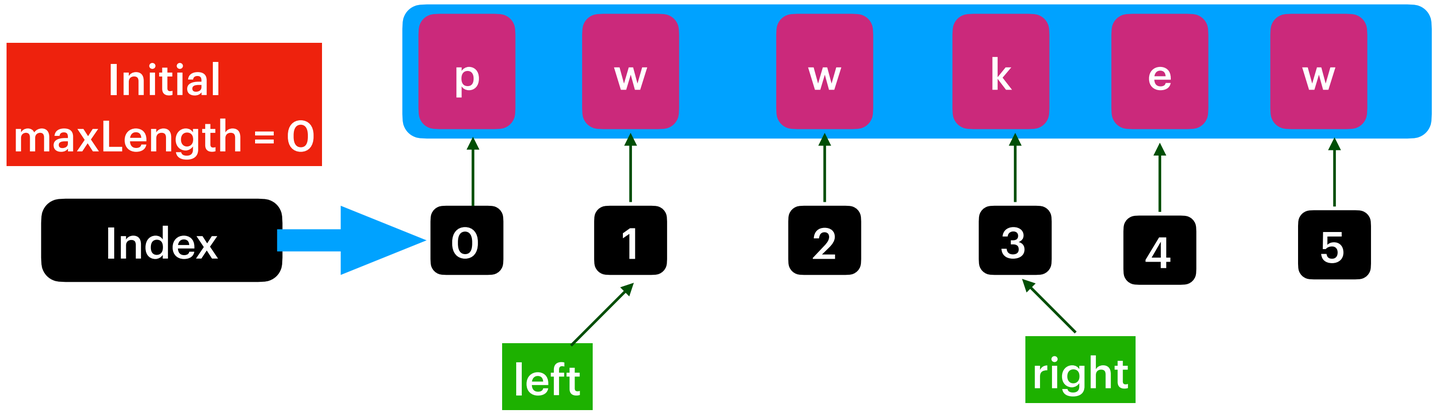
Abstract Solution2 : Maintaining left and right pointer !!!

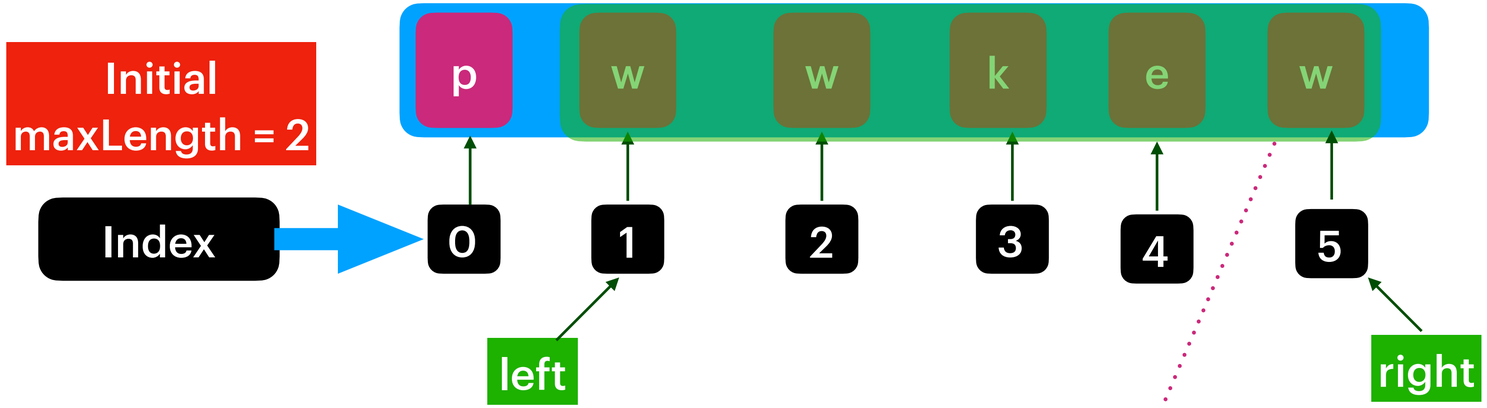




As character repeated in window update values.
max = Math.max(max, right-left)
= Math.max(0, 2-0) = 2
move left pointer one index forward.

Abstract Solution2 : Maintaining left and right pointer !!!





As character repeated in window update values.
max = Math.max(max, right-left)
= Math.max(2, 5-1) = 4

Return max Length : 4 ? Is this right Answer ?

Expected Length : 3