

 Ask a Question

Solution: Valid Word Abbreviation

Let's solve the Valid Word Abbreviation problem using Two Pointers.

We'll cover the following




- Statement
- Solution
 - Time complexity
 - Space complexity

Statement

Given a string `word` and an abbreviation `abbr`, return `TRUE` if the abbreviation matches the given string. Otherwise, return `FALSE`.

A certain word "calendar" can be abbreviated as follows:



- 
- "cal3ar" ("cal end ar" skipping three characters "end" from the word "calendar" still matches the provided abbreviation)
 - "c6r" ("c alendar r" skipping six characters "alendar" from the word "calendar" still matches the provided abbreviation)

The word "internationalization" can also be abbreviated as "i18n" (the abbreviation comes from skipping 18 characters in "internationalization", leaving the first and last letters "i" and "n").

The following are *not* valid abbreviations:

- "c06r" (has leading zeroes)
- "cale0ndar" (replaces an empty string)
- "c24r" ("c al enda r" the replaced substrings are adjacent)

Constraints:

- $1 \leq \text{word.length} \leq 20$
- word consists of only lowercase English letters.
- $1 \leq \text{abbr.length} \leq 10$
- abbr consists of lowercase English letters and digits.
- All the integers in abbr will fit in a 32-bit integer.

Solution

The idea behind this problem is to match each character in the abbreviation, abbr, exactly to the corresponding character in the word. We incrementally iterate over both strings, ensuring they match at each step. To achieve this, we can use the two-pointer technique. We initialize one pointer to the start of the word and the other to the start of the abbreviation.

The two-pointer technique is efficient for solving this problem because it effectively helps in character matching, handling leading zero cases, and skipping the exact number of characters in word as found in abbr.

By maintaining these checks and iterating over both strings, we ensure that the abbreviation correctly represents the word.

Having said this, here's the algorithm that we'll use to solve the given problem:

1. We create two pointers: wordIndex and abbrIndex, both initialized to 0.
2. Next, we iterate through the abbreviations string while abbrIndex is less than the length of abbr:
 - I. If a digit is encountered at abbr[abbrIndex]:
 - i. We then check if that digit is a leading zero. If it is, we return FALSE.
 - ii. Next, we calculate the number from abbr and skip that many characters in word.
 - II. In case the value at index abbr[abbrIndex] is a letter:
 - i. We then check for characters that match with word[wordIndex]. If the characters don't match in both strings, we return FALSE.
 - ii. Next, we increment both wordIndex and abbrIndex by 1.

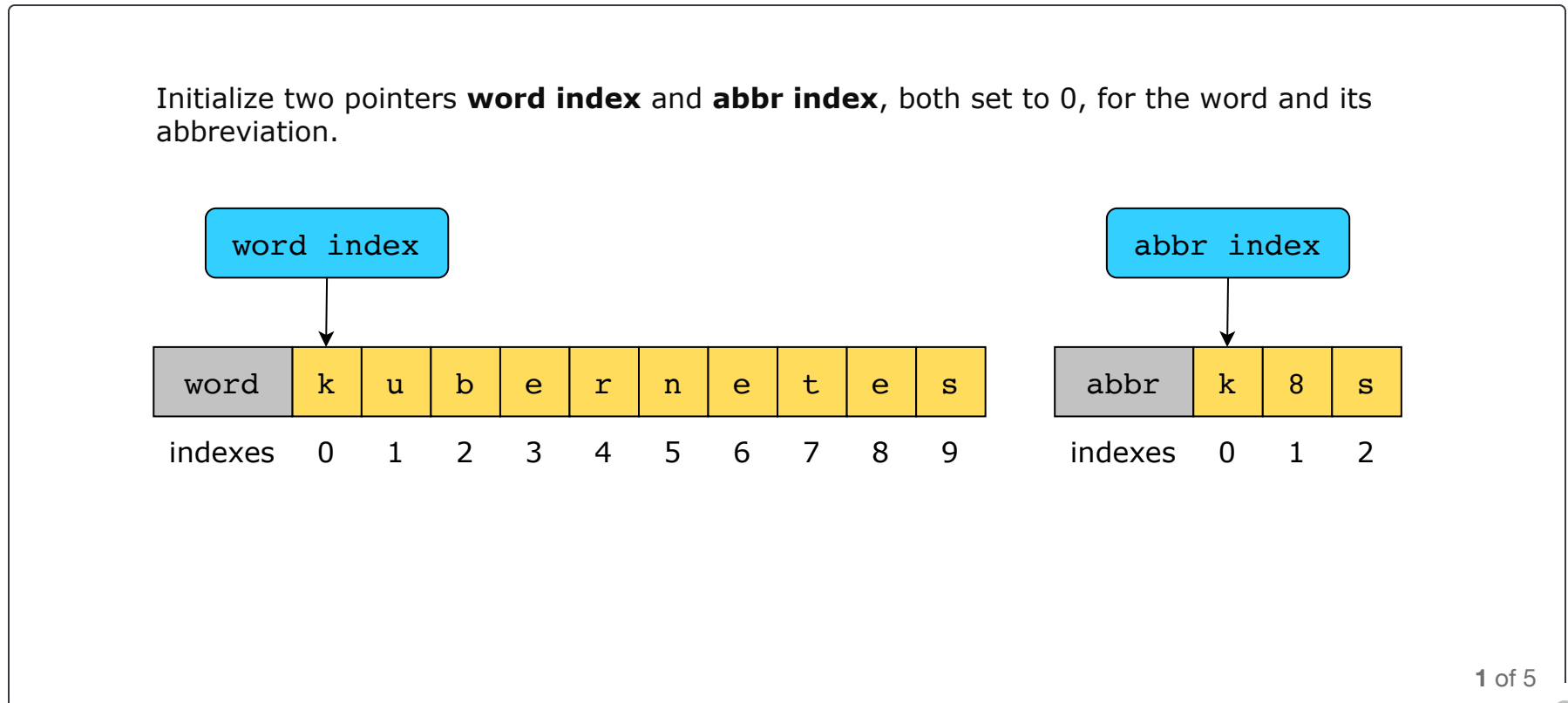
?

Tt

⚙️

3. Finally, we ensure whether both pointers, `wordIndex` and `abbrIndex`, have reached the end of their strings. If they have, we return `TRUE`. Otherwise, we return `FALSE`.

Let's look at the following illustration to get a better understanding of the solution:



Let's look at the code for this solution below:

 Solution Commented Solution

```
class Solution {  
    public static boolean validWordAbbreviation(String word, String abbr) {  
        int wordIndex = 0, abbrIndex = 0;  
  
        while (abbrIndex < abbr.length()) {  
            if (Character.isDigit(abbr.charAt(abbrIndex))) {  
                if (abbr.charAt(abbrIndex) == '0') {  
                    return false;  
                }  
                int num = 0;  
  
                while (abbrIndex < abbr.length() && Character.isDigit(abbr.charAt(abbrIndex))) {  
                    num = num * 10 + (abbr.charAt(abbrIndex) - '0');  
                    abbrIndex++;  
                }  
                wordIndex += num;  
            } else {  
                if (wordIndex >= word.length() || word.charAt(wordIndex) != abbr.charAt(abbrIndex)) {  
                    return false;  
                }  
                wordIndex++;  
                abbrIndex++;  
            }  
        }  
  
        return wordIndex == word.length() && abbrIndex == abbr.length();  
    }  
}
```

Run

 Ask AI Mentor

Save

Reset



Valid Word Abbreviation

Time complexity



The time complexity of the solution above is $O(n)$, where n is the length of the abbreviation string abbr. This is because the solution processes each character of abbr exactly once.

Space complexity

The space complexity is $O(1)$ because the algorithm uses constant extra space regardless of the input size.

[← Back lesson](#)[Mark As Completed](#)[Next →](#)[Valid Word Abbreviation](#)[Strobogrammatic Number](#)