Solution Review: Problem Challenge 2

Comparing Strings containing Backspaces (medium)

Given two strings containing backspaces (identified by the character '#'), check if the two strings are equal.

Example 1:

```
Input: str1="xy#z", str2="xzz#"
Output: true
Explanation: After applying backspaces the strings become "xz" and "xz" respective
ly.
```

Example 2:

```
Input: str1="xy#z", str2="xyz#"
Output: false
Explanation: After applying backspaces the strings become "xz" and "xy" respective
ly.
```

Example 3:

```
Input: str1="xp#", str2="xyz##"
Output: true
Explanation: After applying backspaces the strings become "x" and "x" respectively
.
In "xyz##", the first '#' removes the character 'z' and the second '#' removes th
e character 'y'.
```

Example 4:

```
Input: str1="xywrrmp", str2="xywrrmu#p"
Output: true
Explanation: After applying backspaces the strings become "xywrrmp" and "xywrrmp"
respectively.
```

Solution
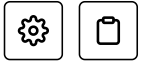
To compare the given strings, first, we need to apply the backspaces. An efficient way to do this would be from the end of both the strings. We can have separate pointers, pointing to the last element of the given strings. We can start comparing the characters pointed out by both the pointers to see if the strings are equal. If, at any stage, the character pointed out by any of the pointers is a backspace ('#'), we will skip and apply the backspace until we have a valid character available for comparison.

Code

Here is what our algorithm will look like:

Java	Python3	C++	JS

```python
1   def backspace_compare(str1, str2):
2     # use two pointer approach to compare the strings
3     index1 = len(str1) - 1
4     index2 = len(str2) - 1
5     while (index1 >= 0 or index2 >= 0):
6       i1 = get_next_valid_char_index(str1, index1)
7       i2 = get_next_valid_char_index(str2, index2)
8       if i1 < 0 and i2 < 0:  # reached the end of both the strings
9         return True
10      if i1 < 0 or i2 < 0:  # reached the end of one of the strings
11        return False
12      if str1[i1] != str2[i2]:  # check if the characters are equal
13        return False
14
15      index1 = i1 - 1
16      index2 = i2 - 1
17
18    return True
19
20
21  def get_next_valid_char_index(str, index):
22    backspace_count = 0
23    while (index >= 0):
24      if str[index] == '#':  # found a backspace character
25        backspace_count += 1
26      elif backspace_count > 0:  # a non-backspace character
27        backspace_count -= 1
28      else:
```

Time complexity

The time complexity of the above algorithm will be $O(M + N)$ where 'M' and 'N' are the lengths of the two input strings respectively.

Space complexity

The algorithm runs in constant space $O(1)$.

Report an
Issue

❓ Ask a Question
(https://discuss.educative.io/tag/solution-review-problem-challenge-2__pattern-two-pointers__grokking-the-
coding-interview-patterns-for-coding-questions)

Report an
Issue

❓ Ask a Question
(https://discuss.educative.io/tag/solution-review-problem-challenge-2__pattern-two-pointers__grokking-the-
coding-interview-patterns-for-coding-questions)