Solution Review: Problem Challenge 2

We'll cover the following

- ^
- Comparing Strings containing Backspaces (medium)
- Solution
 - Code
 - Time complexity
 - Space complexity

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" a

nd "xz" respectively.

Example 2:

Input: str1="xy#z", str2="xyz#"

Output: false

Explanation: After applying backspaces the strings become "xz" a

nd "xy" respectively.

Example 3:







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 the character 'y'.

Example 4:

Input: str1="xywrrmp", str2="xywrrmu#p"

Output: true

Explanation: After applying backspaces the strings become "xywrr

mp" and "xywrrmp" respectively.

Solution

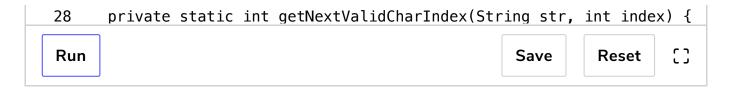
To compare the given strings, first, we need to apply the backspaces. An efficient way to do this would be from the end of both the strings. We can have separate pointers, pointing to the last element of the given strings. We can start comparing the characters pointed out by both the pointers to

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:

```
Python3
                          © C++
👙 Java
                                       JS JS
 1
                                                                          _{\perp}
 2
     class BackspaceCompare {
  3
 4
       public static boolean compare(String str1, String str2) {
 5
         // use two pointer approach to compare the strings
         int index1 = str1.length() - 1, index2 = str2.length() - 1;
  6
 7
         while (index1 \geq= 0 || index2 \geq= 0) {
 8
 9
           int i1 = getNextValidCharIndex(str1, index1);
10
           int i2 = getNextValidCharIndex(str2, index2);
11
12
           if (i1 < 0 && i2 < 0) // reached the end of both the strings
13
             return true;
14
15
           if (i1 < 0 \mid | i2 < 0) // reached the end of one of the strings
             return false;
16
17
18
           if (str1.charAt(i1) != str2.charAt(i2)) // check if the characters a
19
             return false;
20
21
           index1 = i1 - 1;
22
           index2 = i2 - 1;
23
         }
24
25
         return true;
26
       }
27
```



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).

