Solution Review: Problem Challenge 2

We'll cover the following

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" respectively.
```

Example 2:

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

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 the characte
r '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

```

class BackspaceCompare {

  public static boolean compare(String str1, String str2) {
    // use two pointer approach to compare the strings
    int index1 = str1.length() - 1, index2 = str2.length() - 1;
    while (index1 >= 0 || index2 >= 0) {

      int i1 = getNextValidCharIndex(str1, index1);
      int i2 = getNextValidCharIndex(str2, index2);

      if (i1 < 0 && i2 < 0) // reached the end of both the strings
        return true;

      if (i1 < 0 || i2 < 0) // reached the end of one of the strings
        return false;

      if (str1.charAt(i1) != str2.charAt(i2)) // check if the characters are equal
        return false;

      index1 = i1 - 1;
      index2 = i2 - 1;
    }

    return true;
  }

  private static int getNextValidCharIndex(String str, int index) {
```

Run Save Reset

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 $O(1)$.

← Back
Problem Challenge 2

Next →
Problem Challenge 3

✓ Completed

Report an Issue Ask a Question