

Grokking the Coding Interview: Patterns for Coding Questions

12% completed

- Solution Review: Problem Challenge 2**
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: Fast & Slow pointers

- Introduction
- LinkedList Cycle (easy)
- Start of Linked List Cycle (medium)
- Happy Number (medium)
- Middle of the LinkedList (easy)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: Merge Intervals

- Introduction
- Merge Intervals (medium)
- Insert Interval (medium)
- Intervals Intersection (medium)
- Conflicting Appointments (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: Cyclic Sort

- Introduction
- Cyclic Sort (easy)
- Find the Missing Number (easy)
- Find all Missing Numbers (easy)
- Find the Duplicate Number (easy)
- Find all Duplicate Numbers (easy)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: In-place Reversal of a

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" 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 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
1 function backspace_compare(str1, str2) {
2   // use two pointer approach to compare the strings
3   let index1 = str1.length - 1,
4       index2 = str2.length - 1;
5   while (index1 >= 0 || index2 >= 0) {
6     let i1 = get_next_valid_char_index(str1, index1),
7         i2 = get_next_valid_char_index(str2, index2);
8     if (i1 < 0 && i2 < 0) { // reached the end of both the strings
9       return true;
10    }
11    if (i1 < 0 || i2 < 0) { // reached the end of one of the strings
12      return false;
13    }
14    if (str1[i1] !== str2[i2]) { // check if the characters are equal
15      return false;
16    }
17
18    index1 = i1 - 1;
19    index2 = i2 - 1;
20  }
21  return true;
22 }
23
24
25 function get_next_valid_char_index(str, index) {
26   let backspaceCount = 0;
27   while (index >= 0) {
28     if (str[index] === '#') { // found a backspace character
```

MW

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

LinkedList

Introduction

Reverse a LinkedList (easy)

Reverse a Sub-list (medium)

Reverse every K-element Sub-list (medium)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

Pattern: Tree Breadth First Search

Introduction

Binary Tree Level Order Traversal (easy)

Reverse Level Order Traversal (easy)

Zigzag Traversal (medium)

Level Averages in a Binary Tree (easy)

Minimum Depth of a Binary Tree (easy)

Level Order Successor (easy)

Connect Level Order Siblings (medium)

Problem Challenge 1

Solution Review: Problem

RUN

SAVE

RESET

Close

Output

1.805s

true

false

true

true

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

✓ MARK AS COMPLETED

Next →

Problem Challenge 3

Report an Issue