

educative

?

Ask a Question

⚙️

📁

Back To Course Home

Grokking the Coding Interview: Patterns for Coding Questions

16% completed

🔍

Search Course

Solution Review: Problem Challenge 2

Problem Challenge 3

Solution Review: Problem Challenge 3

Pattern: Fast & Slow pointers

Introduction

LinkedList Cycle (easy)

Start of LinkedList Cycle (medium)

Happy Number (medium)

Middle of the LinkedList (easy)

Problem Challenge 1

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

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

We'll cover the following

Comparing Strings containing Backspaces (medium)

Solution

Code

Time complexity

Space complexity

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

Code

Here is what our algorithm will look like:

Java

Python3

C++

JS

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

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

We'll cover the following

Comparing Strings containing Backspaces (medium)

Solution

Code

Time complexity

Space complexity

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

Code

Here is what our algorithm will look like:

Java

Python3

C++

JS

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

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

Space 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.

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

Want to work at Google, Facebook, or Amazon? Get hired faster with anonymous mock interviews conducted by senior engineers from those companies. Detailed feedback helps you prep. [See how](#)

×

← Back

Next →

Problem Challenge 2

Problem Challenge 3

✔️ Mark as Completed

🗨️ Report an Issue